USER MANUAL

DMC-42x0

Manual Rev. 1.0

Galil Motion Control, Inc.

270 Technology Way
Rocklin, California

916.626.0101
support@galilmc.com
galil.com

8/2015

Using This Manual

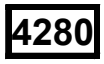
This user manual provides information for proper operation of the DMC-42x0 controller. A separate supplemental manual, the Command Reference, contains a description of the commands available for use with this controller. It is recommended that the user download the latest version of the Command Reference and User Manual from the Galil Website.

http://www.galilmc.com/support/manuals.php

Your DMC-42x0 motion controller has been designed to work with both servo and stepper type motors. Installation and system setup will vary depending upon whether the controller will be used with stepper motors or servo motors. To make finding the appropriate instructions faster and easier, icons will be next to any information that applies exclusively to one type of system. Otherwise, assume that the instructions apply to all types of systems. The icon legend is shown below.

Attention: Pertains to servo motor use.

Attention: Pertains to stepper motor use.



Attention: Pertains to controllers with more than 4 axes.

Please note that many examples are written for the DMC-4240 four-axes controller or the DMC-4280 eight axes controller. Users of the DMC-4230 3-axis controller, DMC-4220 2-axes controller or DMC-4210 1-axis controller should note that the DMC-4230 uses the axes denoted as ABC, the DMC-4220 uses the axes denoted as AB, and the DMC-4210 uses the A-axis only.

Examples for the DMC-4280 denote the axes as A,B,C,D,E,F,G,H. Users of the DMC-4250 5-axes controller. DMC-4260 6-axes controller or DMC-4270, 7-axes controller should note that the DMC-4250 denotes the axes as A,B,C,D,E, the DMC-4260 denotes the axes as A,B,C,D,E,F and the DMC-4270 denotes the axes as A,B,C,D,E,F,G. The axes A,B,C,D may be used interchangeably with X, Y, Z, W.

WARNING	Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machinery. Galil shall not be liable or responsible for any incidental or consequential damages

Contents

Contents iii

Chapter 1 Overview 1

Introduction 1
Part Numbers 2
Electrical Specifications 2
Mechanical Specifications 3
Environmental Specifications 3
Equipment Maintenance 3
Overview of Motor Types 3
Overview of External Amplifiers 4
Functional Elements 5

Chapter 2 Getting Started 8

Layout 8
Dimensions 9
Elements You Need 10
Installing the DMC, Amplifiers, and Motors 11

Chapter 3 Connecting Hardware 24

Overview 24
Overview of Optoisolated Inputs 24
Optoisolated Input Electrical Information 27
TTL Inputs and Outputs 30
Analog Inputs 32
Extended I/O 33
External Amplifier Interface 33

Chapter 4 Software Tools and Communication 41

Introduction 41
Controller Response to Commands 41
Unsolicited Messages Generated by Controller 42
Serial Communication Ports 42
Ethernet Configuration 44
Modbus 46
Data Record 49
GalilSuite (Windows and Linux) 53
Creating Custom Software Interfaces 54

Chapter 5 Command Basics 56

Introduction 56
Command Syntax - ASCII 56
Controller Response to DATA 57
Interrogating the Controller 58

Chapter 6 Programming Motion 60

Overview 60
Independent Axis Positioning 61
Independent Jogging 63
Position Tracking 64
Linear Interpolation Mode 68
Vector Mode: Linear and Circular Interpolation Motion 71
Electronic Gearing 78
Electronic Cam 81
PVT Mode 86
Contour Mode 89
Virtual Axis 93

Stepper Motor Operation....94
Stepper Position Maintenance Mode (SPM)....96
Dual Loop (Auxiliary Encoder)....99
Motion Smoothing....101
Homing....103
High Speed Position Capture (The Latch Function)....105

Chapter 7 Application Programming 106

Overview....106
Program Format....106
Executing Programs - Multitasking....108
Debugging Programs....109
Program Flow Commands....110
Mathematical and Functional Expressions....125
Variables....128
Operands....129
Arrays....130
Input of Data (Numeric and String)....133
Output of Data (Numeric and String)....135
Hardware I/O....139
Extended I/O of the DMC-42x0 Controller....143
Example Applications....145
Using the DMC Editor to Enter Programs....149

Chapter 8 Hardware & Software Protection 151

Introduction....151
Hardware Protection....151
Software Protection....152

Chapter 9 Troubleshooting 156

Overview....156

Chapter 10 Theory of Operation 159

Overview....159
Operation of Closed-Loop Systems....161
System Modeling....162
System Analysis....166
System Design and Compensation....168

Appendices 171

Electrical Specifications....171
Performance Specifications....173
Ordering Options....174
Accessories....178
Input Current Limitations....179
Serial Cable Connections....180
Opto-Isolated Outputs for ICM-2900/ICM-1900/AMP-19520/40....182
Configuring the Amplifier Enable Circuit....183
Signal Descriptions....184
Connectors for DMC-42x0 Main Board....185
A1. ICM-2900 Interconnect Module....192
A2. ICM-2908 Interconnect Module....197
A3. ICM-1900 Interconnect Module....198
A4. AMP-19520/40....202
A5. CB-50-100 Adapter Board....203
A6. CB-50-80 Adapter Board....208
List of Other Publications....212
Training Seminars....212
Contacting Us....213
WARRANTY....214

Integrated Components 215

Overview....215

Chapter 1 Overview

Introduction

The DMC-42x0 Series are Galil's highest performance stand-alone controller. The controller series offers many enhanced features including high speed communications, non-volatile program memory, faster encoder speeds, and improved cabling for EMI reduction.

Each DMC-42x0 provides two communication channels: high speed RS-232 (2 channels up to 115K Baud) and 100 BaseT Ethernet. The controllers allow for high-speed servo control up to 22 million encoder counts/sec and step motor control up to 6 million steps per second. Sample rates as low as 31.25 µsec per axis are available.

A Flash EEPROM provides non-volatile memory for storing application programs, parameters, arrays and firmware. New firmware revisions are easily upgraded in the field.

The DMC-42x0 is available with up to eight axes in a single stand alone unit. The DMC-4210, 4020, 4030, 4040 are one thru four axes controllers and the DMC-4250, 4060, 4070, 4080 are five thru eight axes controllers. All eight axes have the ability to use Galil's integrated amplifiers or drivers and connections for integrating external devices.

Designed to solve complex motion problems, the DMC-42x0 can be used for applications involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, contouring and a PVT Mode. The controller eliminates jerk by programmable acceleration and deceleration with profile smoothing. For smooth following of complex contours, the DMC-42x0 provides continuous vector feed of an infinite number of linear and arc segments. The controller also features electronic gearing with multiple master axes as well as gantry mode operation.

For synchronization with outside events, the DMC-42x0 provides uncommitted I/O, including 8 optoisolated digital inputs (16 inputs for DMC-4250 thru DMC-4280), 8 high power optically isolated outputs (16 outputs for DMC-4250 thru DMC-4280), and 8 analog inputs for interface to joysticks, sensors, and pressure transducers. The DMC-42x0 also has an additional 32 I/O at 3.3V logic. Further I/O is available if the auxiliary encoders are not being used (2 inputs / each axis). Dedicated optoisolated inputs are provided for forward and reverse limits, abort, home, and definable input interrupts.

Commands are sent in ASCII. Additional software is available for automatic-tuning, trajectory viewing on a PC screen, and program development using many environments such as Visual Basic, C, C++ etc. Drivers for Windows XP, Vista and 7 (32 & 64 bit) as well as Linux are available.

Part Numbers

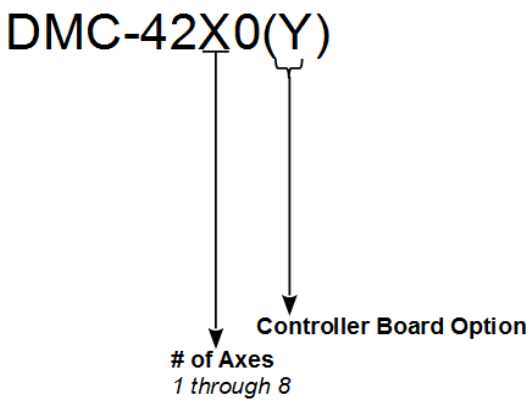


Figure 1.1: Layout of a complete DMC-42x0 part number

DMC, "DMC-42X0(Y)" Options

Option Type	Options	Brief Description	Documentation
X	1,2,3,4,5,6,7, and 8	Number of control axes	N/A
Y	-16bit	16-bit analog inputs	DMC, "DMC-42x0(Y)" Controller Board Options , starting on pg 170.
	4-20mA	4-20mA analog inputs	
	TRES	Encoder terminating resistors	
	SSI	SSI feedback	
	BiSS	BiSS feedback	

Electrical Specifications

Description	Specification
AC Input Voltage	100-240 VAC
AC Input Line Frequency	50-60 Hz
Power Dissipation	12 W

Mechanical Specifications

Description	Specification
Weight	5.2 lb
Length	12.25 in.
Width	5.49 in.
Height	2.37 in.

Environmental Specifications

Description	Specification
Storage Temperature	-25° to +70 ° C
Operating Temperature	0° to +70 ° C
Operating Altitude	10,000 feet

Equipment Maintenance

The DMC-42x0 does not require maintenance.

Overview of Motor Types

The DMC-42x0 can provide the following types of motor control:

1. Standard servo motors with ± 10 volt command signals
2. Brushless servo motors with sinusoidal commutation
3. Step motors with step and direction signals
4. Other actuators such as hydraulics and ceramic motors - For more information, contact Galil.

The user can configure each axis for any combination of motor types, providing maximum flexibility.

Standard Servo Motor with ±10 Volt Command Signal

The DMC-42x0 achieves superior precision through use of a 16-Bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feed-forward, an extra pole filter and integration limits.

The controller is configured by the factory for standard servo motor operation. In this configuration, the controller provides an analog signal (±10 volts) to connect to a servo amplifier. This connection is described in Chapter 2.

Brushless Servo Motor with Sinusoidal Commutation

The DMC-42x0 can provide sinusoidal commutation for brushless motors (BLM). In this configuration, the controller generates two sinusoidal signals for connection with amplifiers specifically designed for this purpose.

Note: The task of generating sinusoidal commutation may be accomplished in the brushless motor amplifier. If the amplifier generates the sinusoidal commutation signals, only a single command signal is required and the controller should be configured for a standard servo motor (described above).

Sinusoidal commutation in the controller can be used with linear and rotary BLMs. However, the motor velocity should be limited such that a magnetic cycle lasts at least 6 milliseconds with a standard update rate of 1 millisecond. For faster motors, please contact the factory.

To simplify the wiring, the controller provides a one-time, automatic set-up procedure. When the controller has been properly configured, the brushless motor parameters may be saved in non-volatile memory.

The DMC-42x0 can control BLMs equipped with Hall sensors as well as without Hall sensors. If Hall sensors are available, once the controller has been setup, the brushless motor parameters may be saved in non-volatile memory. In this case, the controller will automatically estimate the commutation phase upon reset.

This allows the motor to function immediately upon power up. The Hall effect sensors also provide a method for setting the precise commutation phase. Chapter 2 describes the proper connection and procedure for using sinusoidal commutation of brushless motors.

Stepper Motor with Step and Direction Signals

The DMC-42x0 can control stepper motors. In this mode, the controller provides two signals to connect to the stepper motor: Step and Direction. For stepper motor operation, the controller does not require an encoder and operates the stepper motor in an open loop fashion. Chapter 2 describes the proper connection and procedure for using stepper motors.

If encoders are available on the stepper motor, Galil's Stepper Position Maintenance Mode may be used for automatic monitoring and correction of the stepper position. See Stepper Position Maintenance Mode (SPM) in Chapter 6 for more information.

Overview of External Amplifiers

The amplifiers should be suitable for the motor and may be linear or pulse-width-modulated. An amplifier may have current feedback, voltage feedback or velocity feedback.

Amplifiers in Current Mode

Amplifiers in current mode should accept an analog command signal in the ±10 volt range. The amplifier gain should be set such that a +10V command will generate the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V.

Amplifiers in Velocity Mode

For velocity mode amplifiers, a command signal of 10 volts should run the motor at the maximum required speed. The velocity gain should be set such that an input signal of 10V runs the motor at the maximum required speed.

Stepper Motor Amplifiers

For step motors, the amplifiers should accept step and direction signals.

Functional Elements

The DMC-42x0 circuitry can be divided into the following functional groups as shown in Figure 1.2 and discussed below.

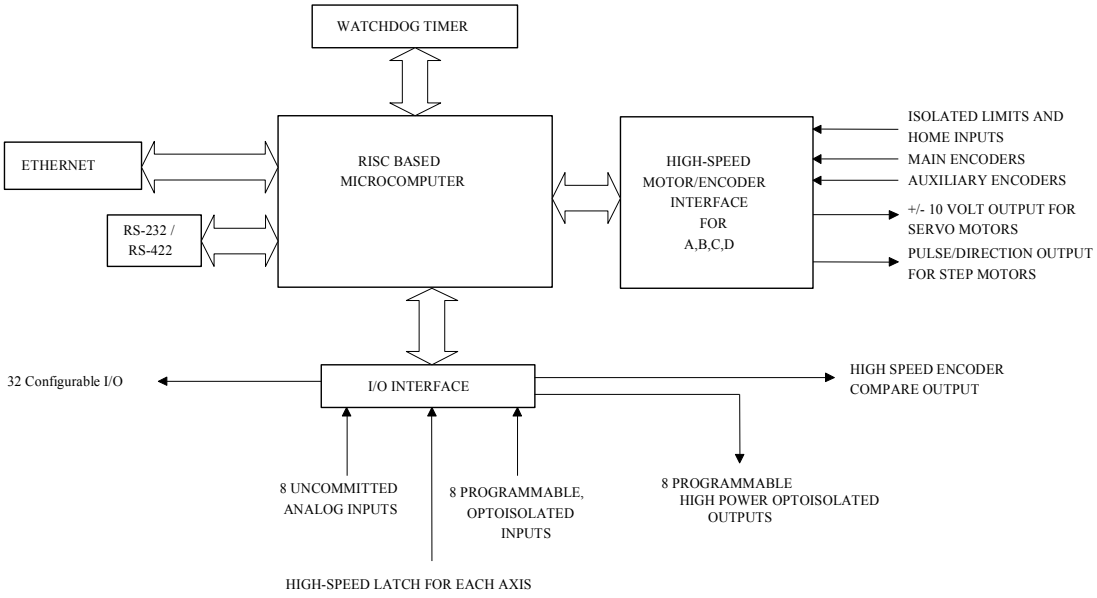


Figure 1.2: DMC-42x0 Functional Elements

Microcomputer Section

The main processing unit of the controller is a specialized Microcomputer with RAM and Flash EEPROM. The RAM provides memory for variables, array elements, and application programs. The flash EEPROM provides non-volatile storage of variables, programs, and arrays. The Flash also contains the firmware of the controller, which is field upgradeable.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 22 MHz. For standard servo operation, the controller generates a ±10 volt analog signal (16-bit DAC). For sinusoidal commutation operation, the controller uses two DACs to generate two ±10 volt analog signals. For stepper motor operation, the controller generates a step and direction signal.

Communication

The communication interface with the DMC-42x0 consists of high speed RS-232 and Ethernet. The Ethernet is 10/100Bt and the two RS-232 channels can generate up to 115K.

General I/O

The DMC-42x0 provides interface circuitry for 8 bi-directional, optoisolated inputs, 8 high power optoisolated outputs and 8 analog inputs with 12-Bit ADC (16-Bit optional). The DMC-42x0 also has an additional 32 I/O (3.3V logic) and unused auxiliary encoder inputs may also be used as additional inputs (2 inputs / each axis). The general inputs as well as the index pulse can also be used as high speed latches for each axis. A high speed encoder compare output is also provided.

4280 The DMC-4250 through DMC-4280 controller provides an additional 8 optoisolated inputs and 8 high power optoisolated outputs.

System Elements

As shown in Figure 1.3, the DMC-42x0 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

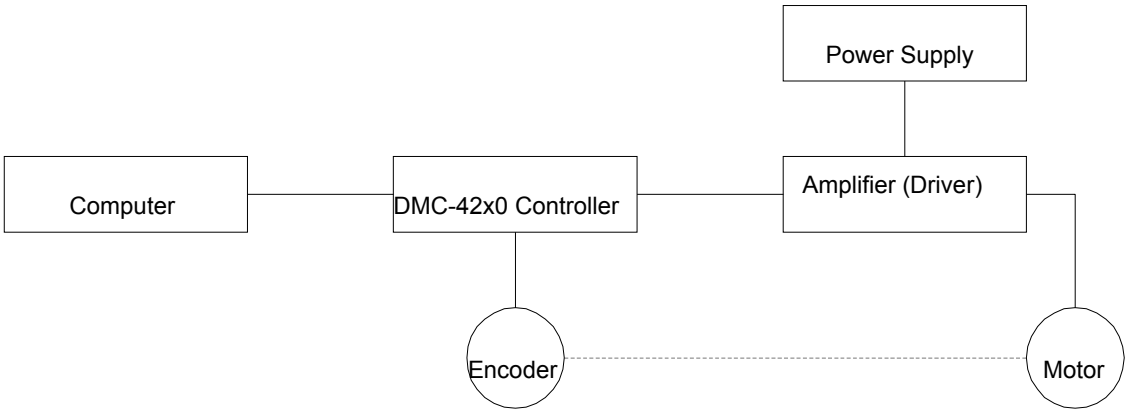


Figure 1.3: Elements of Servo systems

Motor

A motor converts current into torque which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. (Galil's MotorSizer Web tool can help you with motor sizing: www.galilmc.com/support/motorsizer)

The motor may be a step or servo motor and can be brush-type or brushless, rotary or linear. For step motors, the controller can be configured to control full-step, half-step, or microstep drives. An encoder is not required when step motors are used.

Other motors and devices such as Ultrasonic Ceramic motors and voice coils can be controlled with the DMC-42x0.

Amplifier (Driver)

For each axis, the power amplifier converts a ±10 volt signal from the controller into current to drive the motor. For stepper motors, the amplifier converts step and direction signals into current. The amplifier should be sized properly to meet the power requirements of the motor. For brushless motors, an amplifier that provides electronic commutation is required or the controller must be configured to provide sinusoidal commutation. The amplifiers may be either pulse-width-modulated (PWM) or linear. They may also be configured for operation with or without a tachometer. For current amplifiers, the amplifier gain should be set such that a 10 volt command generates the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, 10 volts should run the motor at the maximum speed.

Galil offers amplifiers that are integrated into the same enclosure as the DMC-42x0. See the section in the Appendices or http://galilmc.com/products/accelera/dmc42x0.html for more information.

Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The DMC-42x0 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as MA and MB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (MA+ and MB+) or differential (MA+, MA-, MB+, and MB-). The DMC-42x0 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization.

The DMC-42x0 can be ordered with 120 Ω termination resistors installed on the encoder inputs. See the Ordering Options in the Appendix for more information.

The DMC-42x0 can also interface to encoders with pulse and direction signals. Refer to the “CE” command in the command reference for details.

There is no limit on encoder line density; however, the input frequency to the controller must not exceed 5,500,000 full encoder cycles/second (22,000,000 quadrature counts/sec). For example, if the encoder line density is 10,000 cycles per inch, the maximum speed is 300 inches/second. If higher encoder frequency is required, please consult the factory.

The standard encoder voltage level is TTL (0-5v), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the DMC-42x0. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-42x0 can accept analog feedback (±10v) instead of an encoder for any axis. For more information see the command AF in the command reference.

To interface with other types of position sensors such as absolute encoders, Galil can customize the controller and command set. Please contact Galil to talk to one of our applications engineers about your particular system requirements.

Watch Dog Timer

The DMC-42x0 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AMPEN) which can be used to switch the amplifiers off in the event of a serious DMC-42x0 failure. The AMPEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AMPEN output will go low. The error light will also turn on at this stage. A reset is required to restore the DMC-42x0 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your DMC-42x0 is damaged.

Chapter 2 Getting Started

Layout

DMC-42x0

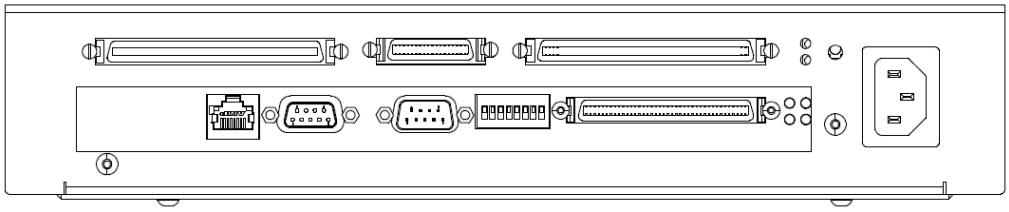


Figure 2.1: Outline of the of the DMC-4280 1-8 axes model

Dimensions

DMC-4240

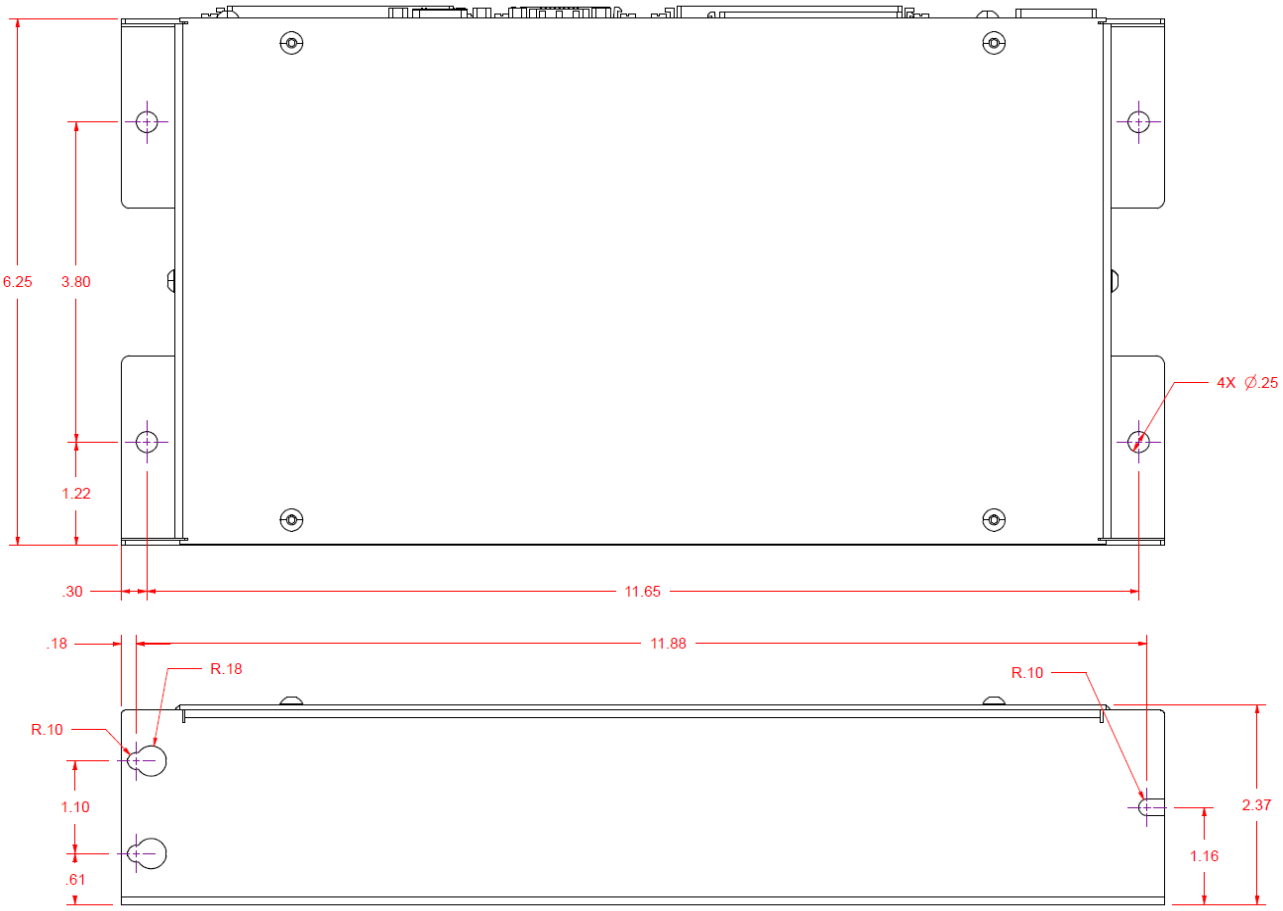


Figure 2.3: Dimensions (in inches) of the DMC-42x0 (Where x= 1-8)

Elements You Need

For a complete system, Galil recommends the following elements:

1. DMC-42x0, motion controller where the x designates number of axis, 1-8.
2. Interconnect.
3. a. (1) ICM-2900 and (1) CABLE-100 for a 1-4 axis DMC-42x0.
   b. (2) ICM-2900s and (2) CABLE-100s for a 5-8 axis DMC-42x0.
   c. An interconnect board provided by the user.
4. Motor Amplifiers.
5. Power Supply for Motor Amplifiers.
6. AC Power source for DMC-42x0 controller.
7. Brush or Brushless Servo motors with Optical Encoders or stepper motors.
   a. Cables for connecting to the DMC-42x0.
8. PC (Personal Computer - RS232 or Ethernet for DMC-42x0)
9. GalilSuite or GalilSuite Lite (Free) software package.

GalilSuite is highly recommended for first time users of the DMC-42x0. It provides step-by-step instructions for system connection, tuning, and analysis.

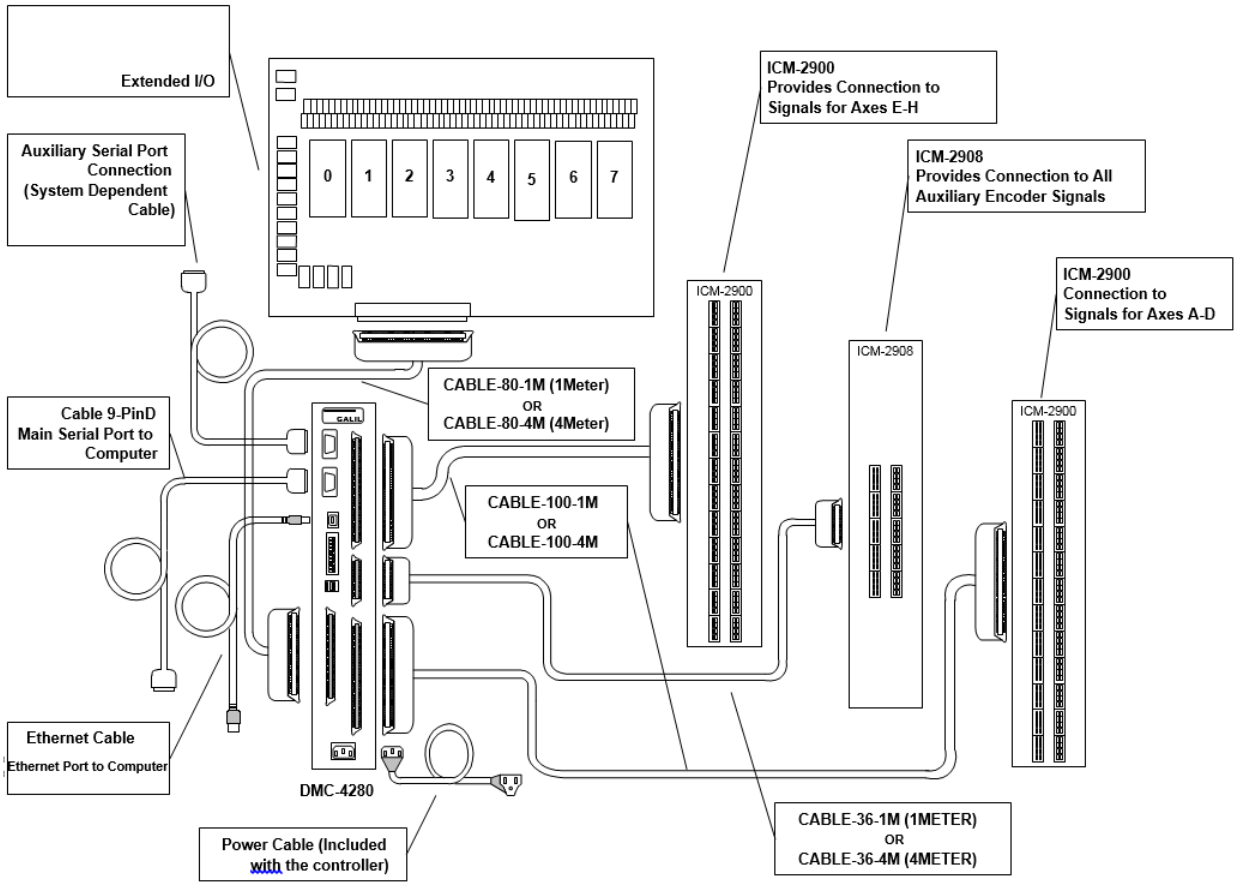


Figure 2.4: Recommended System Elements of DMC-42x0

Installing the DMC, Amplifiers, and Motors

Installation of a complete, operational motion control system consists of the following steps:

Step 1. Determine Overall System Configuration, pg 11

Step 2. Install Jumpers, pg 12

Step 3. Configure Dip Switches ,pg 12

Step 4. Install the Communications Software, pg 13

Step 5. Connect AC Power to the Controller, pg 13

Step 6. Establish Communications with Galil Software, pg 13

Step 7. Connecting Encoder Feedback, pg 13 *Optional for steppers*

Step 8. Setting Safety Features before Wiring Motors, pg 15 *Servo motors only*

Step 9. Wiring Motors to Galil's AMP-19540., pg 17 *Galil amplifiers only*

Step 9a. Commutation of 3-phased Brushless Motors, pg 18 *Brushless motors only*

Step 10. Connecting External Amplifiers and Motors, pg 21 *External amplifiers only*

Step 11. Tune the Servo System, pg 23 *Servo motors only*

WARNING	Electronics are dangerous! Only a certified electrical technician, electrical engineer, or electrical professional should wire the DMC product and related components. Galil shall not be liable or responsible for any incidental or consequential damages. All wiring procedures and suggestions mentioned in the following sections should be done with the controller in a powered-off state. Failing to do so can cause harm to the user or to the controller.

NOTE	The following instructions are given for Galil products only. If wiring an non-Galil device, follow the instructions provided with that product. Galil shall not be liable or responsible for any incidental or consequential damages that occur to a 3rd party device.

Step 1. Determine Overall System Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. The DMC-42x0 can control any combination of brushless motors, brushed motors, and stepper motors. Galil has amplifier options that can drive motors directly but can also control external amplifiers using either a ±10V motor command line or PWM/Step and direction lines. There are also several feedback options that the DMC can accept.

See Part Numbers , pg 2 for understanding your complete DMC unit and part number before continuing.

Step 2. Install Jumpers

Master Reset and Upgrade Jumpers

Jumpers labeled MRST and UPGD are the Master Reset and Upgrade jumpers, respectively.

JP1 on the main board contains two jumpers, MRST and UPGRD. The MRST jumper is the Master Reset jumper. When MRST is connected, the controller will perform a master reset upon PC power up or upon the reset input going low. The MRST can also be set with the DIP switches on the outside of the controller. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in EEPROMwill be ERASED.

The UPGD jumper enables the user to unconditionally update the controller's firmware. This jumper should not be used without first consulting Galil.

Step 3. Configure Dip Switches

Located on the outside of the controller box is a set of 8 DIP switches. Whent eh controller is powered on or reset, the state of the DIP switches are read.

Switch 1 – Master Reset

When this switch is on, the controller will perform a master reset upon PC power up. Whenever the controller has a master reset, all programs and motion control parameters stored in EEPROM will be ERASED. During normal operation, this switch should be off.

Switches 2,3, and 4 are non-functional and reserved

Switches 5 and 6 Main Serial Port Baud Rate

If using the RS232 port for communication, the baud rate is set via switches 5 and 6. To set the baud rate, configure switches 5 and 6as follows:

Switch 5 -19.2	Switch 6 – 38.4	BAUD RATE
ON	OFF	19200
OFF	ON	38400
ON	ON	115200

Switch 7 – Option

When OFF, the controller will use the auto-negotiate function to set the Ethernet connection speed. When the DIP switch is ON, the controller defaults to 10BaseT.

Switch 8 – Motor Off

It is recommended to use the MO switch when connecting motors for the first time. With the MO switch ON, the controller will boot-up in the "motor off" state. In this state the amplifier enable signals are toggled to "inhibit/disable". With this switch OFF

Step 4. Install the Communications Software

After applying power to the controller, a PC is used for programming. Galil's development software enables communication between the controller and the host device. The most recent copy of Galil's development software can be found here:

http://www.galilmc.com/support/software-downloads.php

Step 5. Connect AC Power to the Controller

WARNING	Dangerous voltages, current, temperatures and energy levels exist in this product and the associated amplifiers and servo motor(s). Extreme caution should be exercised in the application of this equipment. Only qualified individuals should attempt to install, set up and operate this equipment. Never open the controller box when DC power is applied

Before applying power, connect the 100-pin cable between the DMC-42x0 and ICM-2900 interconnect module. The DMC-42x0 requires a single AC supply voltage, single phase, 50 Hz or 60 Hz. From 90 VAC to 260 VAC.

The green power light indicator should go on when power is applied.

Step 6. Establish Communications with Galil Software

See Ethernet Configuration, pg 39 for details on using Ethernet with the DMC-42x0. To configure your NIC card using Windows to connect to a DMC controller, see this two-minute video:

http://www.galilmc.com/learning/two-minute-display.php?video=connecting-to-ethernet-controller

For connecting using serial, see RS-232 Configuration, pg 37 for proper configuration of the Main DMC-42x0 serial port.

See the GalilSuite manual for using the software to communicate:

http://www.galilmc.com/support/manuals/galilsuite/index.html

Step 7. Connecting Encoder Feedback

4280 DMC-42x0 controllers with more than 4 axes require a second ICM-2900 and 100-pin cable.

The motion control system typically consists of an ICM-2900 Interface Module, an amplifier for each axis of motion, and a motor to transform the current from the amplifier into torque for motion.

If you are using an ICM-2900, connect it to the DMC-42x0 via the 100-pin high density cable. The ICM-2900 provides screw terminals for access to the connections described in the following discussion.

System connection procedures will depend on system components and motor types. Any combination of motor types can be used with the DMC-42x0. If sinusoidal commutation is to be used, special attention must be paid to the reconfiguration of axes.

The type of feedback the unit is capable of depends on the additional options ordered. Table 2.1 shows the different Encoder feedback types available for the DMC-42x0 including additional options are required. Note that each feedback type has a different configuration command. See the Command Reference for full details on how to properly configure each axis.

Different feedback types can be used on the same controller. For instance, one axis could be using Standard quadrature and the next could be using SSI. By default, all axis are configured for Standard quadrature.

Feedback Type	Configuration Command		Connection Location
Standard quadrature	CE	Standard on all units	Encoder
Step/Dir	CE	Standard on all units	Encoder
Analog[1]	AF	Standard on all units (12-bit Standard. 16-bit optional)	Analog
SSI	SI	Requires the SSI Option, see page(172)	Encoder
BiSS	SS	Requires the BiSS Option, see page(172)	Encoder
None[2]	–	–	--
Other	Contact Galil at 1.800.377.6329		

Table 2.1: Configuration commands, ICM/Part numbers required for a given feedback type

[1] All wiring/electrical information regarding using analog inputs can be found in the Analog Inputs, pg 33.

[2] Although stepper systems do not require feedback, Galil supports a feedback sensor on each stepper axis. Servo motors require a position sensor.

A note about using encoders and steppers:

When a stepper is used, the auxiliary encoder for the corresponding axis is unavailable for an external connection. If an encoder is used for position feedback, connect the encoder to the main encoder input corresponding to that axis. The commanded position of the stepper can be interrogated with TD and the encoder position can be interrogated with TP.

The following steps provide a general guide for connecting encoders to the DMC unit:

Step A. Wire the encoder

The pinouts for the DMC-42x0 Main Board connectors can be found in the Appendix:

Connectors for DMC-42x0 Main Board

The pin-outs and electrical information for SSI and BiSS options can be found here:

DMC-42x0 (BiSS or SSI Option) Pinout, pg. 172.

The encoder pin-outs for Galil Interconnect Modules are listed below.

ICM-2900

A1. ICM-2900 Interconnect Module

ICM-1900

A3. ICM-1900 Interconnect Module

For AMP-19520/40 encoder connections, please refer to the amplifier's User Manual:

http://galilmc.com/download/manual/man_195x0.pdf

Step B. Issue the appropriate configuration commands

Find the appropriate configuration commands for your feedback type as shown in Table 2.1, pg 14.

Step C. Verify proper encoder operation

1. Ensure the motor is off my issuing an MO.
2. Check the current position by issuing TP, the value reported back is in the units of counts.

3. Move the motor by hand and re-issue TP. The returned value should have been incremented or decremented from the first TP. If there is no change, check the encoder wiring and settings and retest starting at Step 1.
4. Using the encoder specification sheet, translate a physical distance of the motor into counts read by the controller. For example, a 2000 line encoder means that the controller reads 2000*4= 8000 counts/revolution and a half turn of the motor would be 4000 counts.
5. Issue TP to determine the current motor position, record this value.
6. Move the motor by hand some measured physical distance.
7. Query TP again. Take the absolute difference from the current TP and the TP recorded from Step 5.
8. Determine if the physical distance moved is equal to the expected amount of counts calculated in Step 4, move on to Step 9. Otherwise, check the encoder wiring and settings and retest starting at Step 1.
9. Perform Step 5-8 again, instead moving a physical distance in the opposite direction. If the physical distance correctly translates to the expected amount of counts, the encoder is wired correctly.

Step D. Reverse encoder direction, if necessary

Table 2.2 below provides instructions for how to reverse the direction of feedback by rewiring the encoder to the DMC controller. The direction of standard, quadrature encoders can be be reversed using the CE command.

NOTE	Reversing direction of the feedback may cause a servo motor to runaway, see Step 8. Setting Safety Features before Wiring Motors, pg 15 regarding Runaway Motors.

Feedback Type		**Directions**
Standard Quadrature	Differential	Swap channels A+ and A-
	Single-ended	Swap channels A+ and B+
SSI or BiSS		Follow encoder manufacturer's instructions.
Analog feedback		Cannot change the direction of feedback without external hardware to invert analog signal.[1]

Table 2.2: Directions for reversing feedback direction based upon feedback type

[1] The polarity of the control loop may still be inverted by either re-wiring the motor or using the MT command, see Step 8. Setting Safety Features before Wiring Motors, pg 15 regarding positive feedback loops.

Step 8. Setting Safety Features before Wiring Motors

This section applies to servo motors only.

Step A. Set Torque Limit

TL will limit the output voltage of the ±10V motor command line. This output voltage is either translated into torque or velocity by the amplifier (Galil's internal amplifiers are in torque mode). This command should be used to avoid excessive torque or speed when initially setting up a servo system. The user is responsible for determining the relationship between the motor command line and the amplifier torque/velocity using the documentation of the motor and/or amplifier.

See the TL setting in the Command Reference for more details.

See the AG command in the command reference for current gains of Galil's internal amplifiers. The amplifier gain can also be used to change the ratio of outputting amps of the amplifier per commanded volts of the controller.

This is another way to limit the amount of current but can also maintain the resolution of the ±10V motor command line.

Step B. Set the Error Limit

When ER (error limit) and OE (off-on-error) is set, the controller will automatically shut down the motors when excess error (|TE| > ER) has occurred. This is an important safety feature during set up as wrong polarity can cause the motor to run away, see **Step C** below for more information regarding runaway motors.

NOTE: Off-on-error (OE) requires the amplifier enable signal to be connected from the controller to the amplifier. This is automatic when using Galil's internal amplifiers, see Step 10. Connecting External Amplifiers and Motors, pg 21 for external amplifiers

Step C. Understanding and Correcting for Runaway Motors

A runaway motor is a condition for which the motor is rotating uncontrollably near it's maximum speed in a single direction. This is often caused by one of two conditions:

1. The amplifier enable signal is the incorrect logic required by the amplifier

   This is only applicable to external amplifiers **only**.

   If the motor is in a MO state when the motor runs away, the MO command is toggling your amplifier "on/enabled" and needs to be reconfigured. The motor is running away because the controller is registering the axis is in an "inactive" and is not attempting to control it's movement. See Step 10. Connecting External Amplifiers and Motors, pg 21 for configuring the amplifier enable signal.

2. The motor and encoder are in opposite polarity causing a positive feedback loop

   Reversed polarity is when a positive voltage on the motor command line results in negative movement of the motor. This will result in a *positive feedback loop* and a runaway motor.

   The following steps can be taken to detect reverse polarity, the A-axis is used as an example:

   1. After connecting your servo motor using either Step 9. Wiring Motors to Galil's AMP-19540., pg 17 or Step 10. Connecting External Amplifiers and Motors, pg 21 issue the following commands:

      ```
      MO A
      KIA= 0
      KPA= 0
      KDA= 0
      SH A
      ```

   2. Check your current position by issuing TP A.
   3. Set a small, positive voltage on your motor command line using the OF command; use a high enough voltage to get the motor to move. This will cause a runaway-like condition so have an appropriate OE set, see **Step B**. Example:

      ```
      OFA= 0.5
      ```

   4. If the motor has not been disabled by OE, disable it by issuing MO A.
   5. Check the position again by using TP A.
   6. If TP has increased, than the motor command line and encoder are in correct polarity. If TP has decreased than the motor command line is in opposite polarity with the encoder.

   If the system has reverse polarity, take the following steps to correct for it:

   Brushed Motor

   Choose *one* of the following:

   1. Reverse the direction of the motor leads by swapping phase A and phase B

2. Reverse the direction of the encoder, see Step 7. Connecting Encoder Feedback, pg 13

Brushless Motor

Choose *one* of the following:

1. Reverse direction of the encoder, see Step 7. Connecting Encoder Feedback, pg 13
2. Reverse direction of the motor by swapping any two motor phases (*or* two hall sensors if using a trapezoidal amplifier). The motor will now have to be re-commutated by using either the Trapezoidal or Sinusoidal method, see Step 9a. Commutation of 3-phased Brushless Motors, pg 18

Non-wiring Options

You can reverse the direction of the motor command line by using the `MT` command *or* reverse direction of the feedback by using the `CE` command (standard quadrature and step/direction feedback only). It is not recommended to correct for polarity using configuration commands as an unexpected condition may arise where these settings are accidentally over-ridden causing a runaway.

See the Command Reference for more details.

Step D. Other Safety Features

This section only provides a brief list of safety features that the DMC can provide. Other features include Encoder Failure Detection (`OA`, `OT`, `OV`) , Automatic Subroutines to create an automated response to events such as limit switches toggling (`#LIMSWI`), command errors (`#POSERR`), and amplifier errors (`TA`, `#AMPERR`), and more. For a full list of features and how to program each see Chapter 8 Hardware & Software Protection, pg 147.

Step 9. Wiring Motors to Galil's AMP-19540.

Table 2.3 below provides a general overview of the connections required for most systems connecting to a Galil amplifier and controller system. Following the table is a step-by-step guide on how to do so.

Motor Type	Required Connections
Brushless servo motor	• Power to controller and internal amplifier • Motor power leads to internal amplifiers • Encoder feedback • Hall sensors (Not required for sinusoidal amplifiers)
Brushed servo motor	• Power to controller and internal amplifier • Motor power leads to internal amplifiers • Encoder feedback

Table 2.3: Synopsis of connections required to connect a motor to Galil's internal amplifiers

Step A. Connect the encoder feedback (*optional for steppers)*

See Step 7. Connecting Encoder Feedback, pg 13.

Step B. Connect the motor power leads and halls (if required) to the Galil amplifier

Table 2.4 lists each of Galil's internal amplifiers and where to find documentation for pin-outs of the amplifier connections and electrical specifications. In addition it describes the commutation method and whether halls are required.

Amplifier	Commutation	Halls Required
AMP-19540	Trapezoidal	Halls required for brushless motors
AMP-19520	Brushed	No

Table 2.4: Amplifier documentation location, commutation, and hall requirements for each internal amplifier.

Pin-outs for the hall signals is found under the ICM being used, refer to Step 7. Connecting Encoder Feedback, pg. 13 for pinouts.

NOTE	**If wiring 3-phased, brushless motors:** ***Skip*** to the additional instructions provided in Step 9a. Commutation of 3-phased Brushless Motors, pg 18 to find proper commutation.

Step C. Issue the appropriate configuration commands

Table 2.5 provides a brief list of configuration commands that may need to be set depending on your motor type and motor specifications.

Command	Description
MT	Configures an axis for use with either a stepper or servo motor
AG	Amplifier gain (A/V for servos or A/Phase for steppers)
BR	Will configure an internal servo amplifier for brushed mode (Also used to ignore halls when the use of external amplifiers is required in lieu of an internal)
AU	Configures the current loop update rate (Can also be used to switch capable amplifiers between chopper and inverter mode)
TL, TK	Limits motor command line output in Volts, thus limiting the current in the amplifier
LC	Configures stepper motor current at holding or “rest” positions

Table 2.5: Sample of motor and amplifier configuration commands

Step D. If using a servo motor, continue to Step 11. Tune the Servo System, pg 23. If using a stepper, continue on to **Step E**.

Step E. Enable and use your motor

A SH will enable the internal amplifier and a MO will disable the internal amplifier. Once enabled, you can send DMC motion commands to move the motor, see Chapter 6 Programming Motion, pg 55 for details.

Step 9a. Commutation of 3-phased Brushless Motors

If a motor is not correctly commutated it will not function as expected. Commutation is the act of properly getting each of the 3 internal phases of a servo motor to switch at the correct time to allow smooth, 360 degree rotation in both directions. The two most common methods for doing so are trapezoidal commutation (use of Hall sensors) and through position sensor algorithms (sinusoidal commutation, no Halls required).

The following sections provide a brief description and guide on how to perform either commutation method including wiring and configuration commands. These sections are divided into Trapezoidal and Sinusoidal:

Trapezoidal Commutation

The AMP-19540 and AMP-19520 support Trapezoidal commutation.

Trapezoidal commutation is a time-tested way for determining the motor location within a magnetic cycle; However, interpretation of hall sensor feedback varies between motor manufactures requiring the user to find the correct wiring combination.

Before wiring the motor the user should determine which is easier: Wiring the hall sensors or wiring the motor phases. This method will start with wiring both the halls and motor phases at random then trying each of the 6 wiring combinations of either the halls *or* the motor phases (not both). For each combination, the user will be asked to check the open-loop velocity in both directions . Some of the wiring combinations will lead to no motion, this is expected. The following directions are given using the A-axis as an example.

1. Wire the 3 motor phase wires and 3 hall sensors randomly. Do not connect the motor to any external mechanics or load, a free spinning motor is required for testing. Take all safety precautions necessary as the motor tests below will result in a runaway condition.

2. Set the PID's and BR to zero and disable off-on-error (OE) to allow for full rotation of the motor in open-loop. Issue the following commands from a Galil terminal program:

```
KPA= 0
KDA= 0
KIA= 0
BRA= 0
OE 0
SH A
```

3. Place a small offset voltage on the motor command line using the OF command (ex OFA= 0.5). The smallest OF possible to see motion is recommended. If no motion presents itself, increase in small increments until you see motion. If your OF is beyond what is expected to see motion, record "no motion" using one of the tables below (Table 2.7 for swapping motor phases or Table 2.8 for swapping halls) and try the next wiring combination.

Note: To stop the motor from spinning use either the MO A command or issue OFA= 0.

4. Once spinning, check the velocity of the motor with the TV A command. Record this value under "+ Velocity" in either Table 2.7 or Table 2.8.

5. Issue an equal but opposite OF. For example, if you previously issued OFA= 0.5 now issue OFA= -0.5. Record this velocity under "- Velocity."

6. Issue OFA= 0 or MO A to stop the motors. Power down the controller and amplifiers system and swap 2 wires of the hall sensors or motor power leads—whichever method is being used (Remember, chose one or the other, not both!). Keep track of what cable combinations have been tested (labeling the phases maybe useful) in the example table in Table 2.6, motor phases were recorded based upon their insulation color.

7. Repeat steps 2-6 for every possible wiring combination, there will be six and Table 2.7 or Table 2.8 below should be completely filled out.

8. The correct wiring combination will be the one with the least difference in magnitude between the velocities in the positive and negative direction. In the case where there are two combinations that meet this criteria, choose the combination that has the higher velocities. In the example table shown in Table 2.6, Trial 1 would be the correct choice.

Trial #	Phase A	Phase B	Phase C	+ Velocity	- Velocity
1	Red	White	Black	153700	-160000
2	Red	Black	White	No motion	No motion
3	White	Black	Red	No motion	No motion
4	White	Red	Black	-141000	139000
5	Black	Red	White	No motion	No motion
6	Black	White	Red	-70000	92000

Table 2.6: Example table showing realistic test results using this commutation method

Trial #	Phase A	Phase B	Phase C	+ Velocity	- Velocity
1					
2					
3					
4					
5					
6					

Table 2.7: Table provided for use with swapping motor phases to achieve trapezoidal communication

Trial #	Hall A	Hall B	Hall C	+ Velocity	- Velocity
1					
2					
3					
4					
5					
6					

Table 2.8: Table provided for use with swapping hall leads to achieve trapezoidal communication

9. Check that the motor phases and encoder feedback are in proper polarity to avoid a runaway condition. Do so by watching the different hall transitions by using the QH command and rotating the motor by hand in an MO state. If the motor and encoder polarity are correct than TP A should report a smaller number when QH A reports 1 than when QH A reports 3. If TP A is larger when QH A reports 1 than 3, then the motor is in a positive feedback state and will runaway when sent movement commands; Reverse the encoder feedback as described in Step 7. Connecting Encoder Feedback, pg 13.

10. Issue MO A and set OFA= 0. Set small, and appropriate values of KP A and KD A and verify the motor holds position once a SH A is issued. The motor is now under closed loop control.

11. Double check commutation by issuing a small jog command (JGA=1000; BG A) and verify the motor spins smoothly for more than 360 degrees. If the user monitors QH during the jog movement it should report a number 1-6 transitioning through the following sequence: 1, 3, 2, 6, 4, 5 and repeating.

12. If no runaway occurs, the motor is ready to be tuned. Skip to Step 11. Tune the Servo System, pg 23.

Step 10. Connecting External Amplifiers and Motors

4280 DMC-42x0 controllers with more than 4 axes require a second ICM-2900 and 100-pin cable.

System connection procedures will depend on system components and motor types. Any combination of motor types can be used with the DMC-42x0. For connecting There can also be a combination of axes running from Galil's AMP-19540/19520 and external amplifiers or drivers.

Table 2.9 below shows a brief synopsis of the connections required, the full step-by-step guide is provided below.

Motor Type	Connection Requirements
Servo motors (Brushed and Brushless)	• Power to controller and amplifier • Amplifier enable • Encoder feedback • Motor command line • See amplifier documentation for motor connections
Stepper motor	• Power to controller and amplifier • Amplifier enable • PWM/Step and direction line • Encoder feedback (optional) • See amplifier documentation for motor connections

Table 2.9: Synopsis of connections required to connect an external amplifier

Step A. Connect the motor to the amplifier

Initially do so *with no connection to the controller*. Consult the amplifier documentation for instructions regarding proper connections. Connect and turn-on the amplifier power supply. If the amplifiers are operating properly, the motor should stand still even when the amplifiers are powered up.

A Note Regarding Commutation

*This section applies to 3-phase external amplifiers **only**.*

External amplifiers often will perform either trapezoidal or sinusoidal commutation without the need of a controller. In this case, be sure to use your amplifiers guide to achieve proper commutation.

Although very rare, if an external amplifier requires the controller to perform sinusoidal commutation, an *additional* ±10 V motor command line may be required from the DMC. Contact Galil if your external amplifier has this requirement.

Step B. Connect the amplifier enable signal

Before making any connections from the amplifier to the controller, verify that the ground level of the amplifier is either floating or at the same potential as earth.

WARNING	When the amplifier ground is not isolated from the power line or when it has a different potential than that of the computer ground, serious damage may result to the computer, controller, and amplifier.

If you are not sure about the potential of the ground levels, connect the two ground signals (amplifier ground and earth) by a 10 kΩ resistor and measure the voltage across the resistor. Only if the voltage is zero, connect the two ground signals directly.

The amplifier enable signal is defaulted to 5V, high amp enable. (the amplifier enable signal will be high when the controller expects the amplifier to be enabled). It is recommended that if an amplifier requires a different configuration, the controller should be be ordered with the desired configuration.

Pin-outs for the amplifier enable signal is found under the ICM being used:

ICM-2900: A1. ICM-2900 Interconnect Module

ICM-1900 :A3. ICM-1900 Interconnect Module

For full electrical specifications and wiring diagrams refer to:
External Amplifier Interface, pg 33

Once the amplifier enable signal is correctly wired , issuing a MO will disable the amplifier and an SH will enable it.

Step C. Connect the Encoders *(optional for stepper systems)*
See Step 7. Connecting Encoder Feedback, pg 13.

Step D. Connect the Command Signals

The DMC-42x0 has two ways of controlling amplifiers:

1. Using a motor command line (±10V analog output)

   The motor and the amplifier may be configured in torque or velocity mode. In the torque mode, the amplifier gain should be such that a 10V signal generates the maximum required current. In the velocity mode, a command signal of 10V should run the motor at the maximum required speed.

2. Using step (0-5V, PWM) and direction (0-5V toggling line), this is referred to as Step/Dir for short.

Pin-outs for the command signals are found under the ICM being used:

The full list of ICM pin-outs are provided in **Step B**, above.

For full electrical specifications refer to:

External Amplifier Interface, pg 33

To configure the command signal type and other configuration commands see Table 2.10 below for a brief synopsis. For a full list of configuration commands see the Command Reference.

Step E. Issue the appropriate configuration Commands

Command	Description
MT	The motor type command configures what type of control method to use (switches axis between motor command or step/dir options)
TL	**Servo only**. Limits the motor command line's continuous output in Volts
TK	**Servo only**. Limits the motor command line's peak output in Volts

Table 2.10: Brief listing of most commonly used configuration commands for the motor command and step/dir lines

Step F. If using a servo motor, continue to Step 11. Tune the Servo System, pg 23. If using a stepper motor, skip to **Step G.**

Step G. Enable and use your motor

A SH will enable the external amplifier, once enabled, you can send DMC motion commands to move the motor, see Chapter 6 Programming Motion, pg 55 for details.

Step 11. Tune the Servo System

Adjusting the tuning parameters is required when using servo motors. A given set of default PID's is provided, but are not optimized and should not be used in practice.

For the theory of operation and a full explanation of all the PID and other filter parameters, see Chapter 10 Theory of Operation, pg 155.

For additional tuning resources and step-by-step tuning guides, see the following:

Application Notes

Manual Tuning Methods: http://www.galil.com/download/application-note/note3413.pdf

Manual Tuning using the Velocity Zone method: http://www.galil.com/download/application-note/note5491.pdf

Autotuning Tools in Galil Suite: http://www.galil.com/download/manual/galilsuite/tuner.html

Chapter 3 Connecting Hardware

Overview

The DMC-42x0 provides optoisolated digital inputs for **forward limit**, **reverse limit, home**, and **abort** signals. The controller also has **8 optoisolated, uncommitted inputs** (for general use) as well as **8 high power optoisolated outputs** and **8 analog inputs** configured for voltages between ±10 volts.

4280 Controllers with 5 or more axes have an additional 8 optoisolated inputs and an additional 8 high power optoisolated outputs.

This chapter describes the inputs and outputs and their proper connection.

Overview of Optoisolated Inputs

Limit Switch Input

The forward limit switch (FLSx) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLSx) inhibits motion in the reverse direction immediately upon activation of the switch. If a limit switch is activated during motion, the controller will make a decelerated stop using the deceleration rate previously set with the SD command. The motor will remain on (in a servo state) after the limit switch has been activated and will hold motor position. The controller can be configured to disable the axis upon the activation of a limit switch, see the OE command in the command reference for further detail.

When a forward or reverse limit switch is activated, the current application program that is running in thread zero will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine which the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch. Automatic Subroutines for Monitoring Conditions are discussed in Chapter 7 Application Programming.

After a limit switch has been activated, further motion in the direction of the limit switch will not be possible until the logic state of the switch returns back to an inactive state. Any attempt at further motion before the logic state has been reset will result in the following error: "`22 - Begin not possible due to limit switch`" error.

The operands, _LFx and _LRx, contain the state of the forward and reverse limit switches, respectively (x represents the axis, A, B, C, D etc.). The value of the operand is either a '0' or '1' corresponding to the logic state of the limit switch. Using a terminal program, the state of a limit switch can be printed to the screen with the command, MG_LFx or MG_LRx. This prints the value of the limit switch operands for the 'x' axis. The logic state of the limit switches can also be interrogated with the TS command. For more details on TS see the Command Reference.

Home Switch Input

Homing inputs are designed to provide mechanical reference points for a motion control application. A transition in the state of a Home input alerts the controller that a particular reference point has been reached by a moving part in the motion control system. A reference point can be a point in space or an encoder index pulse.

The Home input detects any transition in the state of the switch and toggles between logic states 0 and 1 at every transition. A transition in the logic state of the Home input will cause the controller to execute a homing routine specified by the user.

There are three homing routines supported by the DMC-42x0: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FE A, BG A. The Find Edge routine will cause the motor to accelerate, and then slew at constant speed until a transition is detected in the logic state of the Home input. The direction of the FE motion is dependent on the state of the home switch. High level causes forward motion. The motor will then decelerate to a stop. The acceleration rate, deceleration rate and slew speed are specified by the user, prior to the movement, using the commands AC, DC, and SP. *When using the FE command, it is recommended that a high deceleration value be used so the motor will decelerate rapidly after sensing the Home switch.*

The Find Index routine is initiated by the command sequence: FI A, BG A. Find Index will cause the motor to accelerate to the user-defined slew speed (SP) at a rate specified by the user with the AC command and slew until the controller senses a change in the index pulse signal from low to high. The motor then decelerates to a stop at the rate previously specified by the user with the DC command and then moves back to the index pulse and speed HV. Although Find Index is an option for homing, it is not dependent upon a transition in the logic state of the Home input, but instead is dependent upon a transition in the level of the index pulse signal.

The Standard Homing routine is initiated by the sequence of commands HM A, BG A. Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command, DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of HV counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop, moves back to the index, and defines this position as 0. The logic state of the Home input can be interrogated with the command MG_HMA. This command returns a 0 or 1 if the logic state is low or high, respectively. The state of the Home input can also be interrogated indirectly with the TS command.

For examples and further information about Homing, see command HM, FI, FE of the Command Reference and the section entitled Homing in the Programming Motion Section of this manual.

Abort Input

The function of the Abort input is to immediately stop the controller upon transition of the logic state.

NOTE: The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

NOTE: The effect of an Abort input is dependent on the state of the off-on-error function (OE Command) for each axis. If the Off-On-Error function is enabled for any given axis, the motor for that axis will be turned off when the abort signal is generated. This could cause the motor to 'coast' to a stop since it is no longer under servo control. If the Off-On-Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

All motion programs that are currently running are terminated when a transition in the Abort input is detected. This can be configured with the CN command. For information see the Command Reference, OE and CN.

ELO (Electronic Lock-Out) Input

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see in the Appendices. If using a 5-8 axis controller with two integrated amplifiers, the ELO input on the A-D connector should be used. If an ELO is sensed both amplifiers will act on it, and shut down at a hardware level.

Reset Input/Reset Button

When the Reset line is triggered the controller will be reset. The reset line and reset button will not master reset the controller unless the MRST jumper is installed during a controller reset.

Uncommitted Digital Inputs

The DMC-42x0 has 8 optoisolated inputs. These inputs can be read individually using the function @IN[x] where x specifies the input number (1 thru 8). These inputs are uncommitted and can allow the user to create conditional statements related to events external to the controller. For example, the user may wish to have the x-axis motor move 1000 counts in the positive direction when the logic state of DI1 goes high.

The Digital inputs can be used as high speed position latch inputs, see High Speed Position Capture (The Latch Function) for more information.

This can be accomplished by connecting a voltage in the range of +5V to +28V into INCOM of the input circuitry from a separate power supply.

4280

Controllers with more than 4 axes have an additional 8 general optoisolated inputs (inputs 9-16). The INCOM for these inputs is found on the I/O (E-H) D-Sub connector.

An additional 64 I/O are provided at 5V through the extended I/O. These are not optoisolated.

Optoisolated Input Electrical Information

Electrical Specifications

INCOM/LSCOM Max Voltage	24 V_{DC}
INCOM/LSCOM Min Voltage	0 V_{DC}
Minimum current to turn on Inputs	1.2 mA
Minimum current to turn off Inputs once activated (hysteresis)	0.5 mA
Maximum current per input[1]	11 mA
Internal resistance of inputs	2.2 kΩ

[1] See the Input Current Limitations, pg 175 section for more details.

The DMC-42x0's optoisolated inputs are rated to operate with a supply voltage of 5–24 VDC. The optoisolated inputs are powered in banks. For example, INCOM (Bank 0), located on the 44-pin I/O (A-D) D-sub connector, provides power to DI[8:1] (digital inputs), the abort input (ABRT), reset (RST), and electric lock-out (ELO). Table 3.11 shows all the input banks power commons and their corresponding inputs for 1-4 axis controllers and Table 3.12 shows the input banks for 5-8 axis controllers.

Common Signal	Common Signal Location	Powers Inputs Labeled
INCOM (Bank 0)	I/O (A-D) D-Sub Connector	DI[8:1], ABRT, RST, ELO
LSCOM (Bank 0)	I/O (A-D) D-Sub Connector	FLSA, RLSA, HOMA FLSB, RLSB, HOMB FLSC, RLSC, HOMC FLSD, RLSD, HOMD

Table 3.11: 1-4 axis controller INCOM and LSCOM banks and corresponding inputs powered

Common Signal	Common Signal Location	Powers Inputs
INCOM (Bank 0)	I/O (A-D) D-Sub Connector	DI[8:1], ABRT, RST, ELO
LSCOM (Bank 0)	I/O (A-D) D-Sub Connector	FLSA, RLSA, HOMA FLSB, RLSB, HOMB FLSC, RLSC, HOMC FLSD, RLSD, HOMD
INCOM (Bank 1)	I/O (E-H) D-Sub Connector	DI[16:9]
LSCOM (Bank 1)	I/O (E-H) D-Sub Connector	FLSE, RLSE, HOME FLSF, RLSF, HOMF FLSG, RLSG, HOMG FLSH, RLSH, HOMH

Table 3.12: 5-8 axis controller INCOM and LSCOM banks and corresponding inputs powered

The full pin-outs for each bank can be found in the Appendix. For ICM-2900 refer to the A1. ICM-2900 Interconnect Module section on pg. 187. For the ICM-1900 the pin-out can be found in the A3. ICM-1900 Interconnect Module section on, pg. 193.

Wiring the Optoisolated Digital Inputs

To take full advantage of optoisolation, an isolated power supply should be used to provide the voltage at the input common connection. Connecting the ground of the isolated power to the ground of the controller will bypass optoisolation and is not recommended if true optoisolation is desired.

If there is not an isolated supply available, the 5 V_{DC}, 12 V_{DC}, and GND controller references may be used to power INCOM/LSCOM. The current supplied by the controller references are limited, see +5, ±12V Power Output Specifications, pg 168 in the Appendices for electrical specifications. Using the controller reference power completely bypasses optoisolation and is not recommended for most applications.

Banks of inputs can be used as either active high or low. Connecting $+V_s$ to INCOM/LSCOM will configure the inputs for active low as current will flow through the diode when the inputs are pulled to the isolated ground. Connecting the isolated ground to INCOM/LSCOM will configure the inputs for active high as current will flow through the diode when the inputs are pulled up to $+V_s$.

Figure 3.1 - Figure 3.5 are the optoisolated wiring diagrams for powering INCOM/LSCOM (Bank 0) and INCOM/LSCOM (Bank 1) and their corresponding inputs.

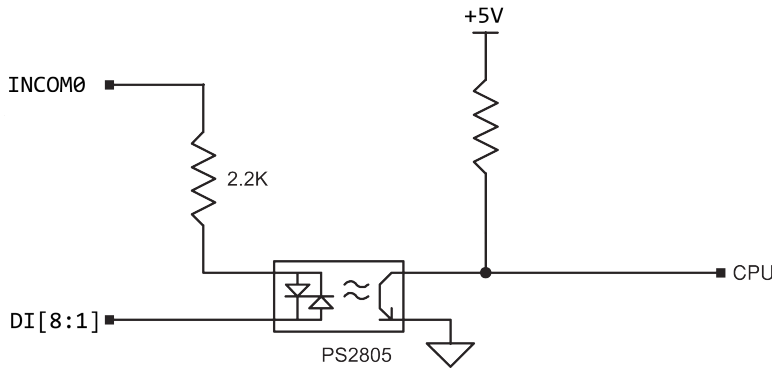


Figure 3.1: Digital Inputs 1-8 (DI[8:1])

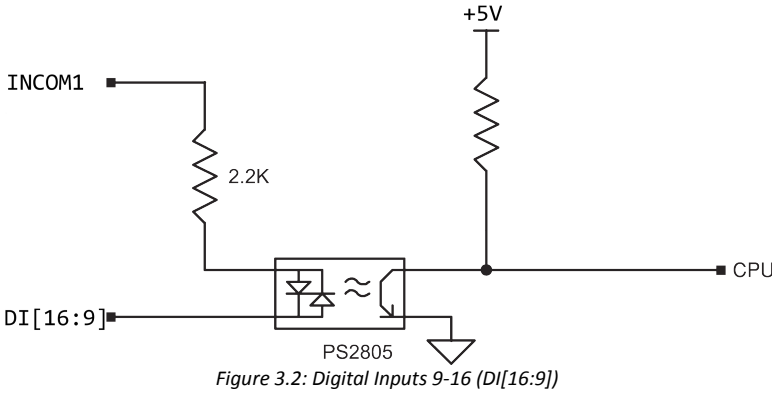


Figure 3.2: Digital Inputs 9-16 (DI[16:9])

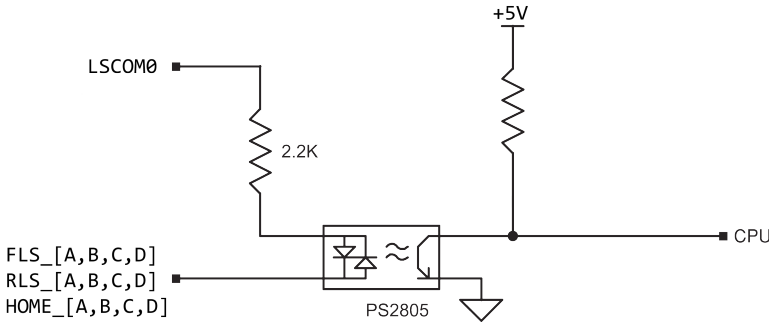


Figure 3.3: Limit Switch Inputs for Axes A-D

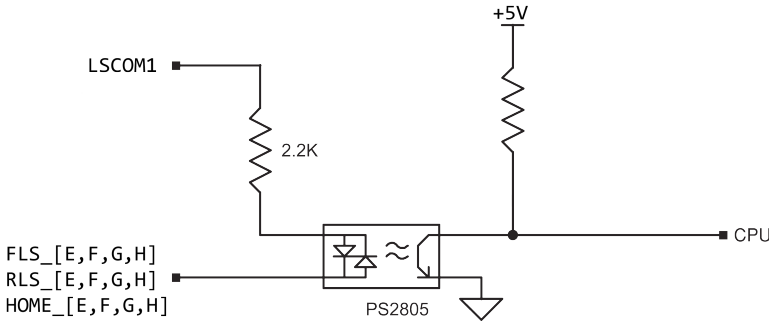


Figure 3.4: Limit Switch Inputs for Axes E-H

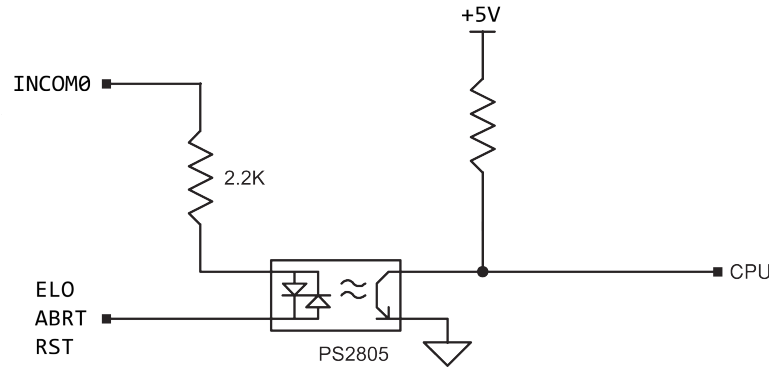


Figure 3.5: ELO, Abort and Reset Inputs

Wiring the Optoisolated Outputs

The output power supply will be connected to Output PWR (labeled OPWR) and the power supply return will be connected to Output GND (labeled ORET). Note that the load is wired between DO and Output GND. The wiring diagram for Bank 0 is shown in Figure 3.6 and Bank 1 in Figure 3.7. See the Opto-Isolated Outputs for ICM-2900/ICM-1900/AMP-19520/40 section in the Appendix for details. Complete pinouts can be found in the respective ICM sections in the appendix: A1. ICM-2900 Interconnect Module or A3. ICM-1900 Interconnect Module.

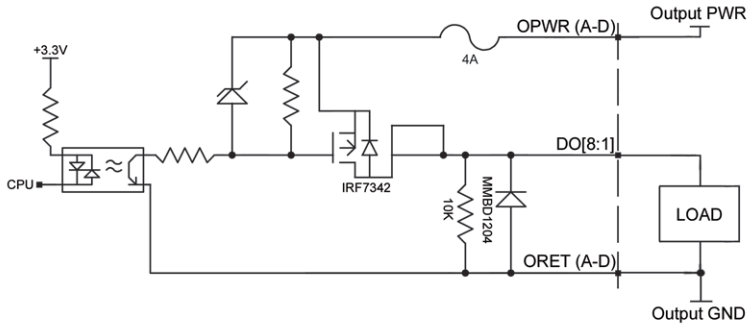


Figure 3.6: 500mA Sourcing wiring diagrams for Bank 0, DO[8:1]

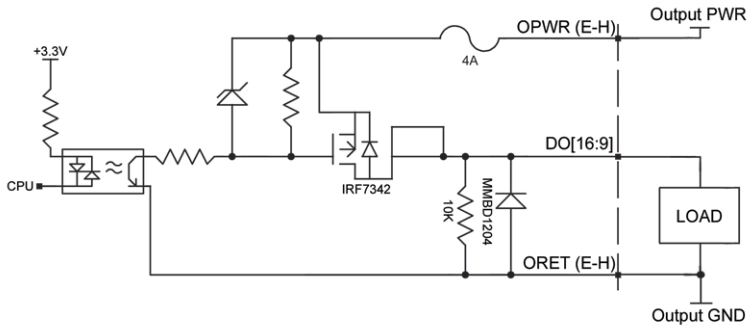


Figure 3.7: 500mA Sourcing wiring diagram for Bank 1, DO[16:9]

TTL Inputs and Outputs

Main Encoder Inputs

The main encoder inputs can be configured for quadrature (default) or pulse and direction inputs. This configuration is set through the CE command. The encoder connections are found on the HD D-sub Encoder connectors and are labeled MA+, MA-, MB+, MB-. The '-' (negative) inputs are the differential inputs to the encoder inputs; if the encoder is a single ended 5V encoder, then the negative input should be left floating. If the encoder is a single ended and outputs a 0-12V signal then the negative input should be tied to the 5V line on the DMC.

When the encoders are setup as step and direction inputs the MA channel will be the step or pulse input, and the MB channel will be the direction input.

The encoder inputs can be ordered with 120 Ω termination resistors installed. See the TRES – Encoder Termination Resistors option in the Appendix for more information.

Electrical Specifications

Maximum Voltage	12 VDC
Minimum Voltage	-12 VDC
Maximum Frequency (Quadrature)	15 MHz

'+' inputs are internally pulled-up to 5V through a 4.7 kΩ resistor

'-' inputs are internally biased to ~1.3V

pulled up to 5V through a 7.1 kΩ resistor

pulled down to GND through a 2.5kΩ resistor

The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. For each axis, the controller has one auxiliary encoder and each auxiliary encoder consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-96. The Aux encoder inputs are not available for any axis that is configured for step and direction outputs (stepper).

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between ± 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 6 volts if the signal is a 0 - 12 volt logic).

Example:

A DMC-4210 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

NOTE: The auxiliary encoder inputs are not available for any axis that is configured for stepper motor.

Electrical Specifications

Maximum Voltage	12 VDC
Minimum Voltage	-12 VDC

'+' inputs are internally pulled-up to 5V through a 4.7kΩ resistor

'-' inputs are internally biased to ~1.3V

pulled up to 5V through a 7.1kΩ resistor

pulled down to GND through a 2.5kΩ resistor

Output Compare

The output compare signal is a TTL ouput signal and is available on the I/O (A-D) D-Sub connector labeled as CMP. An additional output compare signal is available for 5-8 axes controllers on the I/O (E-H) D-sub connector.

Output compare is controlled by the position of any of the main encoder inputs on the controller. The output can be programmed to produce either a brief, active low pulse (250 nsec) based on an incremental encoder value or to activate once ("one shot") when an axis position has been passed. When setup for a one shot, the output will stay low until the OC command is called again. For further information, see the command OC in the Command Reference.

NOTE	Output compare is not valid with sampled feedback types such as: SSI, BiSS, Sin/Cos, and Analog

Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

Error Output

The controller provides a TTL signal, ERR, to indicate a controller error condition. When an error condition occurs, the ERR signal will go low and the controller LED will go on. An error occurs because of one of the following conditions:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

The ERR signal is found on the I/O (A-D) D-Sub connector.

4080 For controllers with 5-8 axes, the ERR signal is duplicated on the I/O (E-H) D-Sub connector.

For additional information see Error Light (Red LED) in Chapter 9 Troubleshooting.

Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

Analog Inputs

The DMC-42x0 has eight analog inputs configured for the range between -10V and 10V. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately .005V. A 16-bit ADC is available as an option (Ex. *DMC-4220(-16bit)-C012-I000*). The analog inputs are specified as AN[x] where x is a number 1 thru 8.

AQ settings

The analog inputs can be set to a range of ±10V, ±5V, 0-5V or 0-10V, this allows for increased resolution when the full ±10V is not required. The inputs can also be set into a differential mode where analog inputs 2,4,6 and 8 can be set to the negative differential inputs for analog inputs 1,3,5 and 7 respectively. See the AQ command in the command reference for more information.

Electrical Specifications

Input Impedance (12 and 16 bit) –

Unipolar (0-5V, 0-10V)	42kΩ
Bipolar (±5V, ±10V)	31kΩ

Extended I/O

The DMC-42x0 controller offers 64 extended TTL I/O points which can be configured as inputs or outputs in 8 bit increments. Configuration is accomplished with command CO – see Extended I/O of the DMC-42x0 Controller The I/O points are accessed through the 80 pin SCSI connector labeled I/O. See the DMC-42x0 Extended I/O 80 Pin High Density Connector section in the Appendix for a complete pin out of the Extended I/O.

Electrical Specifications

Inputs

Max Input Voltage	5.25 VDC
Guarantee High Voltage	2.0 VDC
Guarantee Low Voltage	0.8 VDC

Inputs are internally pulled up to 5V through a 4.7kΩ resistor

Outputs

Sink/Source	20mA

External Amplifier Interface

External Stepper Control

The controller provides step and direction (STPn, DIRn) outputs for every axis available on the controller. These outputs are typically used for interfacing to external stepper drivers, but they can be configured for a PWM output. See the MT command for more details.

PWM/Step and Sign/Direction Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

External Servo Control

The DMC-42x0 command voltage ranges between ±10V and is output on the motor command line - MCMn (where n is A-H). This signal, along with GND, provides the input to the motor amplifiers. The amplifiers must be sized to drive the motors and load. For best performance, the amplifiers should be configured for a torque (current) mode of operation with no additional compensation. The gain should be set such that a 10 volt input results in the maximum required current.

Motor Command Line Electrical Specifications

Output Voltage	±10 VDC
Motor Command Output Impedance	500 Ω

Amplifier Enable

The DMC-42x0 has an amplifier enable signal - AENn (where n is A-H) which changes under the following conditions:

- The motor-off command MO is given or the watchdog timer activates.
- The OE command (Enable Off-On-Error) is set and the position error exceeds the error limit (set by ER).
- A limit switch is reached (see OE command in the Command Reference for more information).

The default configuration of the amplifier enable signal is 5V active high amp enable (HAEN) sinking. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed by configuring the Amplifier Enable Circuit on the ICM-2900 or ICM-1900.

If your amplifier requires a different configuration than the default 5V HAEN sinking it is highly recommended that the DMC-42x0 is ordered with the desired configuration. Contact Galil for more information on ordering different configurations.

Note: Many amplifiers designate the enable input as 'inhibit'.

Configuring the Amplifier Enable Circuit

Configuring the Amplifier Enable for ICM-2900/ICM-1900

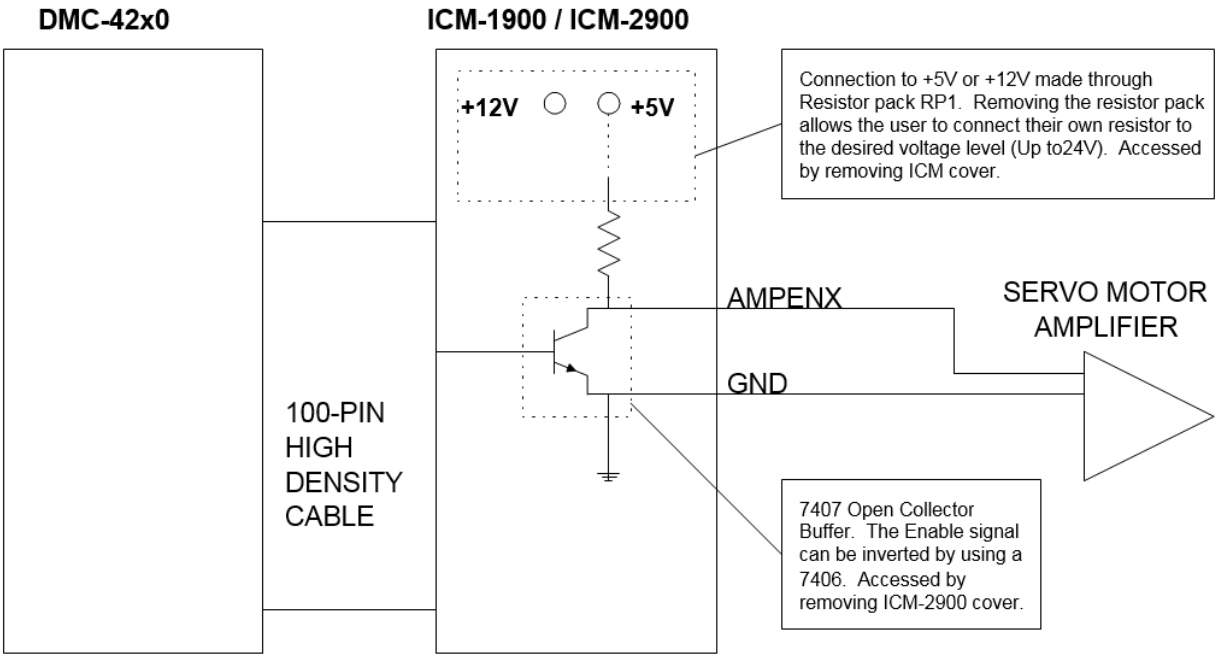
The ICM-1900 and ICM-2900 modules can be configured to provide an active low signal to enable external amplifiers. These modules can also be configured for voltage levels other than TTL.

-LAEN Option:

The standard configuration of the AEN signal is TTL active high. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity can be changed when using a Galil Interconnect Module. To change the polarity from active high (5 volts = enable, zero volts = disable) to active low (zero volts = enable, 5 volts = disable), replace the socketed IC, 7407, with a 7406. These IC's are labeled U6 on the ICM-1900 and U2 on the ICM-2900 and can be accessed by removing the cover. This option can be requested when ordering the unit by specifying the -LAEN option.

-Changing the Amplifier Enable Voltage Level:

To change the voltage level of the AEN signal, note the state of the resistor pack, labeled RP1 on the ICM-1900 / ICM-2900. When Pin 1 is on the 5V mark, the output voltage is 0-5V. To change to 12 volts, pull the resistor pack and rotate it so that Pin 1 is on the 12 volt side. If you remove the resistor pack, the output signal is an open collector, allowing the user to connect an external supply with voltages up to 24V.


DMC-42x0
ICM-1900 / ICM-2900
+12V
+5V
Connection to +5V or +12V made through Resistor pack RP1. Removing the resistor pack allows the user to connect their own resistor to the desired voltage level (Up to24V). Accessed by removing ICM cover.
AMPENX
SERVO MOTOR AMPLIFIER
GND
100-PIN HIGH DENSITY CABLE
7407 Open Collector Buffer. The Enable signal can be inverted by using a 7406. Accessed by removing ICM-2900 cover.

Chapter 4 Software Tools and Communication

Introduction

The default configuration DMC-42x0, with the default CMB-41012 communication board, has two RS232 ports and 1 Ethernet port. An additional Ethernet port is available with the CMB-41022. The main RS-232 port is the data set and can be configured through the jumpers on the top of the controller. The auxiliary RS-232 port is the data term and can be configured with the software command CC. This configuration can be saved using the Burn (BN) instruction. The Ethernet port is a 10/100BASE-T connection that auto-negotiates the speed and half or full duplex.

The GalilTools software package is available for PC computers running Microsoft Windows® to communicate with the DMC-42x0 controller. This software package has been developed to operate under Windows and Linux, and include all the necessary drivers to communicate to the controller. In addition, GalilTools includes a software development communication library which allows users to create their own application interfaces using programming environments such as C, C++, Visual Basic, and LabVIEW.

The following sections in this chapter are a description of the communications protocol, and a brief introduction to the software tools and communication techniques used by Galil. At the application level, GalilTools is the basic programs that the majority of users will need to communicate with the controller, to perform basic setup, and to develop application code (.dmc programs) that is downloaded to the controller. At the Galil API level, the GalilTools Communication Library is available for users who wish to develop their own custom application programs to communicate to the controller. Custom application programs can utilize API function calls directly to our DLL's. At the driver level, we provide fundamental hardware interface information for users who desire to create their own drivers.

Controller Response to Commands

Most DMC-42x0 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return. Multiple commands may be concatenated by inserting a semicolon between each command.

Instructions are sent in ASCII, and the DMC-42x0 decodes each ASCII character (one byte) one at a time. It takes approximately 40 µsec for the controller to decode each command.

After the instruction is decoded, the DMC-42x0 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or the controller will respond with a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the main RS-232 port back through the RS-232 port, and to commands which are sent via the Ethernet port back through the Ethernet port.

For instructions that return data, such as Tell Position (TP), the DMC-42x0 will return the data followed by a carriage return, line feed and : .

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided to enable associating the DMC-42x0 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the main RS-232 port or Ethernet port. This response could be generated as a result of messages using the MG command OR as a result of a command error. These responses are known as unsolicited messages since they are not generated as the direct response to a command.

Messages can be directed to a specific port using the specific Port arguments – see the MG and CF commands in the Command Reference. If the port is not explicitly given or the default is not changed with the CF command, unsolicited messages will be sent to the default port. The default port is the main serial port. When communicating via an Ethernet connection, the unsolicited messages must be sent through a handle that is not the main communication handle from the host. The GalilTools software automatically establishes this second communication handle.

The controller has a special command, CW, which can affect the format of unsolicited messages. This command is used by Galil Software to differentiate response from the command line and unsolicited messages. The command, CW1 causes the controller to set the high bit of ASCII characters to 1 of all unsolicited characters. This may cause characters to appear garbled to some terminals. This function can be disabled by issuing the command, CW2. For more information, see the CW command in the Command Reference.

When handshaking is used (hardware and/or software handshaking) characters which are generated by the controller are placed in a FIFO buffer before they are sent out of the controller. The size of the RS-232 buffer is 512 bytes. When this buffer becomes full, the controller must either stop executing commands or ignore additional characters generated for output. The command CW,1 causes the controller to ignore all output from the controller while the FIFO is full. The command, CW ,0 causes the controller to stop executing new commands until more room is made available in the FIFO. This command can be very useful when hardware handshaking is being used and the communication line between controller and terminal will be disconnected. In this case, characters will continue to build up in the controller until the FIFO is full. For more information, see the CW command in the Command Reference.

Serial Communication Ports

The RS-232 and RS-422 (optional) are located on the CMB (communication board) of the DMC-42x0. Note that the auxiliary port is essentially the same as the main port except inputs and outputs are reversed.

RS-232 Configuration

The pin-outs for the RS-232 ports can be found in the Appendix in the RS-422-Main Port section, pg. 185.

Configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS232 communication can be selected by setting the proper switch configuration on the front panel according to the table below.

Baud Rate Selection

JP1 JUMPER SETTINGS		BAUD RATE
19.2	38.4	
ON	OFF	19200
OFF	ON	38400
OFF	OFF	115200

Handshaking

The RS232 main port is set for hardware handshaking. Hardware Handshaking uses the RTS and CTS lines. The CTS line will go high whenever the DMC-42x0 is not ready to receive additional characters. The RTS line will inhibit the DMC-42x0 from sending additional characters. Note, the RTS line goes high for inhibit.

Auxiliary RS-232 Port Configuration

The main purpose of the auxiliary RS232 port is to connect to external devices that cannot use DMC code to communicate. It is important to note that the Aux port is not an interpreted port and cannot receive DMC Galil commands directly. Instead, use CI, #COMINT, and the P2 operands to handle received data on this port.

NOTE: If you are connecting the RS-232 auxiliary port to a terminal or any device which is a DATASET, it is necessary to use a connector adapter, which changes a dataset to a dataterm. This cable is also known as a 'null' modem cable.

CC Command

The CC, or Configure Communications command, configures the auxiliary ports properties including: Baud rate, handshaking, enable/disabled port, and echo. See the CC command in the Command Reference for a full description and command syntax.

If the CC command is configured for hardware handshaking it is required to use the RTS and CTS lines. The RTS line will go high whenever the DMC is not ready to receive additional characters. The CTS line will inhibit the DMC from sending additional characters. Note, the CTS line goes high for inhibit.

RS-422 Configuration

The DMC-42x0 can be ordered with the main and/or auxiliary port configured for RS-422 communication. RS-422 communication is a differentially driven serial communication protocol that should be used when long distance serial communication is required in an application.

See RS-422-Main Port section, pg. 185 for pin-outs and details of the RS-422 options.

Ethernet Configuration

Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The DMC-42x0 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master, or client, connects to the slave, or server, through a series of packet handshakes in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgment is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP/IP does not require a "connection". If information is lost, the controller does not return a colon or question mark. Because UDP does not provide for lost information, the sender must re-send the packet.

It is recommended that the motion control network containing the controller and any other related devices be placed on a “closed” network. If this recommendation is followed, UDP/IP communication to the controller may be utilized instead of a TCP connection. With UDP there is less overhead, resulting in higher throughput. Also, there is no need to reconnect to the controller with a UDP connection. Because handshaking is built into the Galil communication protocol through the use of colon or question mark responses to commands sent to the controller, the TCP handshaking is not required.

Packets must be limited to 512 data bytes (including UDP/TCP IP Header) or less. Larger packets could cause the controller to lose communication.

NOTE: In order not to lose information in transit, the user must wait for the controller's response before sending the next packet.

Addressing

There are three levels of addresses that define Ethernet devices. The first is the MAC or hardware address. This is a unique and permanent 6 byte number. No other device will have the same MAC address. The DMC-42x0 MAC address is set by the factory and the last two bytes of the address are the serial number of the board. To find the Ethernet MAC address for a DMC-42x0 unit, use the TH command. A sample is shown here with a unit that has a serial number of 3:

Sample MAC Ethernet Address: 00-50-4C-20-04-AF

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number that usually looks like this: 192.168.15.1. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the DMC-42x0 controller can be done in a number of ways.

The first method for setting the IP address is using a DHCP server. The DH command controls whether the DMC-42x0 controller will get an IP address from the DHCP server. If the unit is set to DH1 (default) and there is a DHCP server on the network, the controller will be dynamically assigned an IP address from the server. Setting the board to DH0 will prevent the controller from being assigned an IP address from the server.

The second method to assign an IP address is to use the BOOT-P utility via the Ethernet connection. The BOOT-P functionality is only enabled when DH is set to 0. Either a BOOT-P server on the internal network or the Galil software may be used. When opening the Galil Software, it will respond with a list of all DMC-42x0’s and other controllers on the network that do not currently have IP addresses. The user must select the board and the software will assign the specified IP address to it. This address will be burned into the controller (BN) internally to save the IP address to the non-volatile memory.

NOTE: if multiple boards are on the network – use the serial numbers to differentiate them.

CAUTION	Be sure that there is only one BOOT-P or DHCP server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the DMC-42x0 controller upon linking it to the network. In order to ensure that the IP address is correct, please contact your system administrator before connecting the I/O board to the Ethernet network.

The third method for setting an IP address is to send the IA command through the RS-232 port. (Note: The IA command is only valid if DH0 is set). The IP address may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number (e.g. IA 124,51,29,31 or IA 2083724575). Type in BN to save the IP address to the DMC-42x0 non-volatile memory.

NOTE: Galil strongly recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world.

The third level of Ethernet addressing is the UDP or TCP port number. The Galil board does not require a specific port number. The port number is established by the client or master each time it connects to the DMC-42x0 board. Typical port numbers for applications are:

Port 23: Telnet
Port 502: Modbus

Communicating with Multiple Devices

The DMC-42x0 is capable of supporting multiple masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

NOTE: The term "Master" is equivalent to the internet "client". The term "Slave" is equivalent to the internet "server".

An Ethernet handle is a communication resource within a device. The DMC-42x0 can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARPs do not occupy handles.) If all 8 handles are in use and a 9^{th} master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

NOTE: There are a number of ways to reset the controller. Hardware reset (push reset button or power down controller) and software resets (through Ethernet or RS232 by entering RS).

When the Galil controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a signed 32 bit number. A port number may also be specified, but if it is not, it will default to 1000. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time. Otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<179>2 This will open handle #2 and connect to the IP address 151.25.255.9, port 179, using TCP/IP)

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port with the CF command). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the z axis position to handle #6.)

Multicasting

A multicast may only be used in UDP/IP and is similar to a broadcast (where everyone on the network gets the information) but specific to a group. In other words, all devices within a specified group will receive the information that is sent in a multicast. There can be many multicast groups on a network and are differentiated by their multicast IP address. To communicate with all the devices in a specific multicast group, the information can be sent to the multicast IP address rather than to each individual device IP address. All Galil controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

Using Third Party Software

Galil supports DHCP, ARP, BOOT-P, and Ping which are utilities for establishing Ethernet connections. DHCP is a protocol used by networked devices (clients) to obtain the parameters necessary for operation in an Internet Protocol network. ARP is an application that determines the Ethernet (hardware) address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. Ping is used to check the communication between the device at a specific IP address and the host computer.

The DMC-42x0 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

Modbus

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The DMC-42x0 can use a specific slave address or default to the handle number. The port number for Modbus is 502.

The Modbus protocol has a set of commands called function codes. The DMC-42x0 supports the 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)
06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The DMC-42x0 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

MBh = -1,len,array[] where len is the number of bytes

array[] is the array with the data

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information refer to the Command Reference.

The third level of Modbus communication uses standard Galil commands. Once the slave has been configured, the commands that may be used are @IN[], @AN[], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

I/O Number = (HandleNum*1000) + ((Module-1)*4) + (BitNum-1)

Where HandleNum is the handle number from 1 (A) to 8 (H). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

Modbus Examples

Example #1

DMC-4240 connected as a Modbus master to a RIO-47120 via Modbus. The DMC-4240 will set or clear all 16 of the RIO's digital outputs

1. Begin by opening a connection to the RIO which in our example has IP address 192.168.1.120

   ```
   IHB=192,168,1,120<502>2          (Issued to DMC-4240)
   ```

2. Dimension an array to store the commanded values. Set array element 0 equal to 170 and array element 1 equal to 85. (array element 1 configures digital outputs 15-8 and array element 0 configures digital outputs 7-0)

   ```
   DM myarray[2]
   myarray[0] = 170          (which is 10101010 in binary)
   myarray[1] = 85           (which is 01010101in binary)
   ```

3. a) Send the appropriate MB command. Use function code 15. Start at output 0 and set/clear all 16 outputs based on the data in myarray[]

   ```
   MBB=,15,0,16,myarray[]
   ```

3. b) Set the outputs using the SB command.

   ```
   SB2001;SB2003;SB2005;SB2007;SB2008;SB2010;SB2012;SB2014;
   ```

Results:

Both steps 3a and 3b will result in outputs being activated as below. The only difference being that step 3a will set and clear all 16 bits where as step 3b will only set the specified bits and will have no affect on the others.

Bit Number	Status
0	0
1	1
2	0
3	1
4	0
5	1
6	0
7	1

Bit Number	Status
8	1
9	0
10	1
11	0
12	1
13	0
14	1
15	0

Example #2

DMC-4240 connected as a Modbus master to a 3rd party PLC. The DMC-4240 will read the value of analog inputs 3 and 4 on the PLC located at addresses 40006 and 40008 respectively. The PLC stores values as 32-bit floating point numbers which is common.

1. Begin by opening a connection to the PLC which has an IP address of 192.168.1.10 in our example

```
IHB=192,168,1,10<502>2
```

2. Dimension an array to store the results

```
DM myanalog[4]
```

3. Send the appropriate MB command. Use function code 4 (as specified per the PLC). Start at address 40006. Retrieve 4 modbus registers (2 modbus registers per 1 analog input, as specified by the PLC)

```
MBB=,4,40006,4,myanalog[]
```

Results:

Array elements 0 and 1 will make up the 32 bit floating point value for analog input 3 on the PLC and array elements 2 and 3 will combine for the value of analog input 4.

```
myanalog[0]=16412=0x401C
myanalog[1]=52429=0xCCCD
myanalog[2]=49347=0xC0C3
myanalog[3]=13107=0x3333
```

Analog input 3 = 0x401CCCCD = 2.45V

Analog input 4 = 0xC0C33333 = -6.1V

Example #3

DMC-4240 connected as a Modbus master to a hydraulic pump. The DMC-4240 will set the pump pressure by writing to an analog output on the pump located at Modbus address 30000 and consisting of 2 Modbus registers forming a 32 bit floating point value.

1. Begin by opening a connection to the pump which has an IP address of 192.168.1.100 in our example

```
IHB=192,168,1,100<502>2
```

2. Dimension and fill an array with values that will be written to the PLC

```
DM pump[2]
pump[0]=16531=0x4093
pump[1]=13107=0x3333
```

3. Send the appropriate MB command. Use function code 16. Start at address 30000 and write to 2 registers using the data in the array pump[]

```
MBB=,16,30000,2,pump[]
```

Results:

Analog output will be set to 0x40933333 which is 4.6V

Data Record

The DMC-42x0 can provide a binary block of status information with the use of the QR and DR commands. These commands, along with the QZ command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments:

```
QR ABCDEFGHST
```

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes.

Data Record Map Key	
Acronym	**Meaning**
UB	Unsigned byte
UW	Unsigned word
SW	Signed word
SL	Single long record
UL	Unsigned long

General Controller Information and Status

ADDR	TYPE	ITEM	ADDR	TYPE	ITEM
00	UB	1st Byte of Header	30-31	SW	Reserved
01	UB	2nd Byte of Header	32-33	SW	Reserved
02	UB	3rd Byte of Header	34-35	SW	Reserved
03	UB	4th Byte of Header	36-37	SW	Reserved
04-05	UW	sample number	38-39	SW	Reserved
06	UB	general input block 0 (inputs 1-8)	40-41	SW	Reserved
07	UB	general input block 1 (inputs 9-16)	42	UB	Ethernet Handle A Status
08	UB	general input block 2 (inputs 17-24)	43	UB	Ethernet Handle B Status
09	UB	general input block 3 (inputs 25-32)	44	UB	Ethernet Handle C Status
10	UB	general input block 4 (inputs 33-40)	45	UB	Ethernet Handle D Status
11	UB	general input block 5 (inputs 41-48)	46	UB	Ethernet Handle E Status
12	UB	general input block 6 (inputs 49-56)	47	UB	Ethernet Handle F Status
13	UB	general input block 7 (inputs 57-64)	48	UB	Ethernet Handle G Status
14	UB	general input block 8 (inputs 65-72)	49	UB	Ethernet Handle H Status
15	UB	general input block 9 (inputs 73-80)	50	UB	error code
16	UB	general output block 0 (outputs 1-8)	51	UB	thread status – see bit field map below
17	UB	general output block 1 (outputs 9-16)	52-55	UL	Amplifier Status
18	UB	general output block 2 (outputs 17-24)	56-59	UL	Segment Count for Contour Mode
19	UB	general output block 3 (outputs 25-32)	60-61	UW	Buffer space remaining – Contour Mode
20	UB	general output block 4 (outputs 33-40)	62-63	UW	segment count of coordinated move for S plane
21	UB	general output block 5 (outputs 41-48)	64-65	UW	below
22	UB	general output block 6 (outputs 49-56)	66-69	SL	distance traveled in coordinated move for S plane
23	UB	general output block 7 (outputs 57-64)	70-71	UW	Buffer space remaining – S Plane
24	UB	general output block 8 (outputs 65-72)	72-73	UW	segment count of coordinated move for T plane
25	UB	general output block 9 (outputs 73-80)	74-75	UW	Coordinated move status for T plane – see bit field map
26-27	SW	Reserved	76-79	SL	distance traveled in coordinated move for T plane
28-29	SW	Reserved	80-81	UW	Buffer space remaining – T Plane

Axis Information

ADDR	TYPE	ITEM	ADDR	TYPE	ITEM
82-83	UW	A axis status – see bit field map below	226-227	UW	E axis status – see bit field map below
84	UB	A axis switches – see bit field map below	228	UB	E axis switches – see bit field map below
85	UB	A axis stop code	229	UB	E axis stop code
86-89	SL	A axis reference position	230-233	SL	E axis reference position
90-93	SL	A axis motor position	234-237	SL	E axis motor position
94-97	SL	A axis position error	238-241	SL	E axis position error
98-101	SL	A axis auxiliary position	242-245	SL	E axis auxiliary position
102-105	SL	A axis velocity	246-249	SL	E axis velocity
106-109	SL	A axis torque	250-253	SL	E axis torque
110-111	SW or UW [1]	A axis analog input	254-255	SW or UW [1]	E axis analog input
112	UB	A Hall Input Status	256	UB	E Hall Input Status
113	UB	Reserved	257	UB	Reserved
114-117	SL	A User defined variable (ZA)	258-261	SL	E User defined variable (ZE)
118-119	UW	B axis status – see bit field map below	262-263	UW	F axis status – see bit field map below
120	UB	B axis switches – see bit field map below	264	UB	F axis switches – see bit field map below
121	UB	B axis stop code	265	UB	F axis stop code
122-125	SL	B axis reference position	266-269	SL	F axis reference position
126-129	SL	B axis motor position	270-273	SL	F axis motor position
130-133	SL	B axis position error	274-277	SL	F axis position error
134-137	SL	B axis auxiliary position	278-281	SL	F axis auxiliary position
138-141	SL	B axis velocity	282-285	SL	F axis velocity
142-145	SL	B axis torque	286-289	SL	F axis torque
146-147	SW or UW [1]	B axis analog input	290-291	SW or UW [1]	F axis analog input
148	UB	B Hall Input Status	292	UB	F Hall Input Status
149	UB	Reserved	293	UB	Reserved
150-153	SL	B User defined variable (ZB)	294-297	SL	F User defined variable (ZF)
154-155	UW	C axis status – see bit field map below	298-299	UW	G axis status – see bit field map below
156	UB	C axis switches – see bit field map below	300	UB	G axis switches – see bit field map below
157	UB	C axis stop code	301	UB	G axis stop code
158-161	SL	C axis reference position	302-305	SL	G axis reference position
162-165	SL	C axis motor position	306-309	SL	G axis motor position
166-169	SL	C axis position error	310-313	SL	G axis position error
170-173	SL	C axis auxiliary position	314-317	SL	G axis auxiliary position
174-177	SL	C axis velocity	318-321	SL	G axis velocity
178-181	SL	C axis torque	322-325	SL	G axis torque
182-183	SW or UW [1]	C axis analog input	326-327	SW or UW [1]	G axis analog input
184	UB	C Hall Input Status	328	UB	G Hall Input Status
185	UB	Reserved	329	UB	Reserved
186-189	SL	C User defined variable (ZC)	330-333	SL	G User defined variable (ZG)
190-191	UW	D axis status – see bit field map below	334-335	UW	H axis status – see bit field map below
192	UB	D axis switches – see bit field map below	336	UB	H axis switches – see bit field map below
193	UB	D axis stop code	337	UB	H axis stop code
194-197	SL	D axis reference position	338-341	SL	H axis reference position
198-201	SL	D axis motor position	342-345	SL	H axis motor position
202-205	SL	D axis position error	346-349	SL	H axis position error
206-209	SL	D axis auxiliary position	350-353	SL	H axis auxiliary position
210-213	SL	D axis velocity	354-357	SL	H axis velocity
214-217	SL	D axis torque	358-361	SL	H axis torque
218-219	SW or UW [1]	D axis analog input	362-363	SW or UW [1]	H axis analog input
220	UB	D Hall Input Status	364	UB	H Hall Input Status
221	UB	Reserved	365	UB	Reserved
222-225	SL	D User defined variable (ZD)	366-369	SL	H User defined variable (ZH)

[1] Will be either a Signed Word or Unsigned Word depending upon AQ setting. See AQ in the Command Reference for more information.

Data Record Bit Field Maps

Header Information - Byte 0, 1 of Header:

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
1	N/A	N/A	N/A	N/A	I Block Present in Data Record	T Block Present in Data Record	S Block Present in Data Record
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
H Block Present in Data Record	G Block Present in Data Record	F Block Present in Data Record	E Block Present in Data Record	D Block Present in Data Record	C Block Present in Data Record	B Block Present in Data Record	A Block Present in Data Record

Bytes 2, 3 of Header:

Bytes 2 and 3 make a word which represents the Number of bytes in the data record, including the header.

Byte 2 is the low byte and byte 3 is the high byte

NOTE: The header information of the data records is formatted in little endian (reversed network byte order).

Thread Status (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Thread 7 Running	Thread 6 Running	Thread 5 Running	Thread 4 Running	Thread 3 Running	Thread 2 Running	Thread 1 Running	Thread 0 Running

Coordinated Motion Status for S or T Plane (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	N/A	N/A	N/A	N/A	N/A	N/A	N/A

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	Motion is slewing	Motion is stopping due to ST or Limit Switch	Motion is making final deceleration	N/A	N/A	N/A

Axis Status (1 Word)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	Mode of Motion PA or PR	Mode of Motion PA only	(FE) Find Edge in Progress	Home (HM) in Progress	1st Phase of HM complete	2nd Phase of HM complete or FI command issued	Mode of Motion Coord. Motion

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion is stopping due to ST of Limit Switch	Motion is making final deceleration	Latch is armed	3rd Phase of HM in Progress	Motor Off

Axis Switches (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Latch Occurred	State of Latch Input	N/A	N/A	State of Forward Limit	State of Reverse Limit	State of Home Input	Stepper Mode

Amplifier Status (4 Bytes)

BIT 31	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24
N/A	N/A	N/A	N/A	N/A	N/A	ELO Active (Axis E-H)	ELO Active (Axis A-D)

BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16
Peak Current H-axis	Peak Current G-axis	Peak Current F-axis	Peak Current E-axis	Peak Current D-axis	Peak Current C-axis	Peak Current B-axis	Peak current A-axis

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Hall Error H-axis	Hall Error G-axis	Hall Error F-axis	Hall Error E-axis	Hall Error D-axis	Hall Error C-axis	Hall Error B-axis	Hall Error A-axis

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Under Voltage Axis (E-H)	Over Temp. Axis (E-H)	Over Voltage Axis (E-H)	Over Current Axis (E-H)	Under Voltage Axis (A-D)	Over Temp. Axis (A-D)	Over Voltage Axis (A-D)	Over Current Axis (A-D)

Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the command TV (Tell Velocity). See command reference for more information about TV.

The Torque information is represented as a number in the range of ±32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

QZ Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

BYTE #	INFORMATION
0	Number of axes present
1	number of bytes in general block of data record
2	number of bytes in coordinate plane block of data record
3	Number of Bytes in each axis block of data record

GalilSuite (Windows and Linux)

GalilSuite is Galil's latest set of development tools for the latest generation of Galil controllers. It is highly recommended for all first-time purchases of Galil controllers as it provides easy set-up, tuning and analysis. GalilSuite replaces GalilToolS with an improved user-interface, real-time scopes, advanced tuning methods, and communications utilities.

Supported Controllers

- DMC42x0
- DMC41x3
- DMC30010
- DMC21x3/2
- RIO47xxx
- DMC18x6 - PCI Driver required, separate installer
- DMC18x0 - PCI Driver required, separate installer
- DMC18x2* - PCI Driver required, separate installer

Contact Galil for other hardware products

Supported Operating Systems**

- Microsoft Windows 8
- Microsoft Windows 7
- Microsoft Windows XP SP3
- Scope, Watch, and Viewer support require an Ethernet or PCI connection and controller firmware supporting the DR command

* No Scope, Watch, or Viewer support.
** Contact Galil for other OS options.

The GalilSuitecontains the following tools:

Tool	Description
Launcher	Launcher Tool with the ability to create custom profiles to manage controller connections
Terminal	For sending and receiving Galil commands
Editor	To easily create and work on multiple Galil programs simultaneously
Viewer	To see a complete status of all controllers on a single screen
Scope	For viewing and manipulating data for multiple controllers real-time
Watch	For simplified debugging of any controller on the system and a display of I/O and motion status
Tuner	With up to four methods for automatic and manual PID tuning of servo systems
Configuration	For modifying controller settings, backup/restore and firmware download

The latest version of GalilSuite can be downloaded here:

http://www.galilmc.com/support/software-downloads.php_

For information on using GalilSuite see the user manual:

http://www.galilmc.com/support/manuals.php

Creating Custom Software Interfaces

Galil provides programming tools so that users can develop their own custom software interfaces to a Galil controller. For new applications, Galil recommends the GalilTools Communication Libraries.

HelloGalil – Quick Start to PC programming

For programmers developing Windows applications that communicate with a Galil controller, the HelloGalil library of quick start projects immediately gets you communicating with the controller from the programming language of your choice. In the "Hello World" tradition, each project contains the bare minimum code to demonstrate communication to the controller and simply prints the controller's model and serial numbers to the screen Figure 4.1.

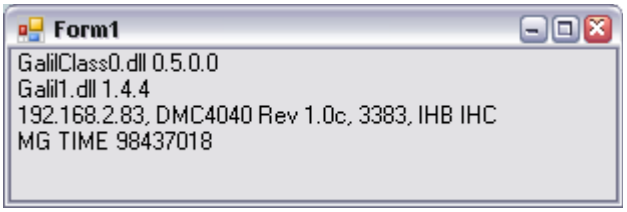


Figure 4.1: Sample program output

http://www.galilmc.com/support/hello_galil.html

Galil Communication Libraries

The Galil Communication Library (Galil class) provides methods for communication with a Galil motion controller over Ethernet, RS-232 or PCI buses. It consists of a native C++ Library and a similar COM interface which extends compatibility to Windows programming languages (e.g. VB, C#, etc).

A Galil object (usually referred to in sample code as "g") represents a single connection to a Galil controller.

For Ethernet controllers, which support more than one connection, multiple objects may be used to communicate with the controller. An example of multiple objects is one Galil object containing a TCP handle to a DMC-42x0 for commands and responses, and one Galil object containing a UDP handle for unsolicited messages from the controller. If recordsStart() is used to begin the automatic data record function, the library will open an additional UDP handle to the controller (transparent to the user).

The library is conceptually divided into six categories:

1. Connecting and Disconnecting - functions to establish and discontinue communication with a controller.
2. Basic Communication - The most heavily used functions for command-and-response and unsolicited messages.
3. Programs - Downloading and uploading embedded programs.
4. Arrays - Downloading and uploading array data.
5. Advanced - Lesser-used calls.
6. Data Record - Access to the data record in both synchronous and asynchronous modes.

C++ Library (Windows and Linux)

Both Full and Lite versions of GalilTools ship with a native C++ communication library. The Linux version (libGalil.so) is compatible with g++ and the Windows version (Galil1.dll) with Visual C++ 2008. Contact Galil if another version of the C++ library is required. See the getting started guide and the hello.cpp example in /lib.

COM (Windows)

To further extend the language compatibility on Windows, a COM (Component Object Model) class built on top of the C++ library is also provided with Windows releases. This COM wrapper can be used in any language and IDE supporting COM (Visual Studio 2005, 2008, etc). The COM wrapper includes all of the functionality of the base C++ class. See the getting started guide and the hello.* examples in \lib for more info.

For more information on the GalilTools Communications Library, see the online user manual.

http://www.galilmc.com/support/manuals/galiltools/library.html

Chapter 5 Command Basics

Introduction

The DMC-42x0 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands are sent in ASCII.

The DMC-42x0 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion.

Commands can be sent "live" over the communications port for immediate execution by the DMC-42x0, or an entire group of commands can be downloaded into the DMC-42x0 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter.

This section describes the DMC-42x0 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-42x0 instructions is included in the *Command Reference*.

Command Syntax - ASCII

DMC-42x0 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the DMC-42x0 command interpreter.

NOTE: If you are using a Galil terminal program, commands will not be processed until an <return> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <return> command.

NOTE	All DMC commands are two-letters sent in upper case!

For example, the command

```
PR 4000 <return>            Position relative
```

Implicit Notation

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The <return> terminates the instruction. The space between PR and 4000 is optional.

For specifying data for the A,B,C and D axes, commas are used to separate the axes. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained.

To view the current values for each command, type the command followed by a ? for each axis requested.

`PR 1000`	Specify A only as 1000
`PR ,2000`	Specify B only as 2000
`PR ,,3000`	Specify C only as 3000
`PR ,,,4000`	Specify D only as 4000
`PR 2000, 4000,6000, 8000`	Specify A,B,C and D
`PR ,8000,,9000`	Specify B and D only
`PR ?,?,?,?`	Request A,B,C,D values
`PR ,?`	Request B value only

Explicit Notation

The DMC-42x0 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as A, B, C or D. An equals sign is used to assign data to that axis. For example:

`PRA=1000`	Specify a position relative movement for the A axis of 1000
`ACB=200000`	Specify acceleration for the B axis as 200000

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST AB stops motion on both the A and B axes. Commas are not required in this case since the particular axis is specified by the appropriate letter A, B, C or D. If no parameters follow the instruction, action will take place on all axes. Here are some examples of syntax for requesting action:

`BG A`	Begin A only
`BG B`	Begin B only
`BG ABCD`	Begin all axes
`BG BD`	Begin B and D only
`BG`	Begin all axes

4080

For controllers with 5 or more axes, the axes are referred to as A,B,C,D,E,F,G,H. The specifiers X,Y,Z,W and A,B,C,D may be used interchangeably.

`BG ABCDEFGH`	Begin all axes
`BG D`	Begin D only

Coordinated Motion with more than 1 axis

When requesting action for coordinated motion, the letter S or T is used to specify the coordinated motion. This allows for coordinated motion to be setup for two separate coordinate systems. Refer to the CA command in the Command Reference for more information on specifying a coordinate system. For example:

`BG S`	Begin coordinated sequence, S
`BG TD`	Begin coordinated sequence, T, and D axis

Controller Response to DATA

The DMC-42x0 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-42x0 will return a ?.

`:bg`	invalid command, lower case
`?`	DMC-42x0 returns a ?

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code type the command TC1. For example:

?TC1	Tell Code command
1 Unrecognized command	Returned response

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete listing of all codes is listed in the TC command in the Command Reference section.

Interrogating the Controller

Interrogation Commands

The DMC-42x0 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 Application Programming and the Command Reference.

Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R^V	Firmware Revision Information
SC	Stop Code
TA	Tell Amplifier Error
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TI	Tell Input
TP	Tell Position
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity

For example, the following example illustrates how to display the current position of the X axis:

TP A	Tell position A
0	Controllers Response
TP AB	Tell position A and B
0,0	Controllers Response

Interrogating Current Commanded Values.

Most commands can be interrogated by using a question mark (?) as the axis specifier. Type the command followed by a ? for each axis requested.

PR ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The controller can also be interrogated with operands.

Operands

Most DMC-42x0 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

MG 'operand' where 'operand' is a valid DMC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the A axis can be assigned to the variable 'V' with the command:

V=_TPA

The Command Reference denotes all commands which have an equivalent operand as "Operand Usage". Also, see description of operands in Chapter 7 Application Programming.

Command Summary

For a complete command summary, see *Command Reference* manual.

http://www.galilmc.com/support/manuals.php

Chapter 6 Programming Motion

Overview

The DMC-42x0 provides several modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

The DMC-4210 are single axis controllers and use X-axis motion only. Likewise, the DMC-4220 use X and Y, the DMC-4230 use X,Y, and Z, and the DMC-4240 use X,Y,Z, and W. The DMC-4250 use A,B,C,D, and E. The DMC-4260 use A,B,C,D,E, and F. The DMC-4270 use A,B,C,D,E,F, and G. The DMC-4280 use the axes A,B,C,D,E,F,G, and H.

The example applications described below will help guide you to the appropriate mode of motion.

4080 For controllers with 5 or more axes, the specifiers, ABCDEFGH, are used. XYZ and W may be interchanged with ABCD.

EXAMPLE APPLICATION	MODE OF MOTION	COMMANDS
Absolute or relative positioning where each axis is independent and follows prescribed velocity profile.	Independent Axis Positioning	PA, PR, SP, AC, DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG, AC, DC, ST
Absolute positioning mode where absolute position targets may be sent to the controller while the axis is in motion.	Position Tracking	PA, AC, DC, SP, PT
Motion Path described as incremental position points versus time.	Contour Mode	CM, CD, DT
Motion Path described as incremental position, velocity and delta time	PVT Mode	PV, BT
2 to 8 axis coordinated motion where path is described by linear segments.	Linear Interpolation Mode	LM, LI, LE, VS,VR, VA, VD
2-D motion path consisting of arc segments and linear segments, such as engraving or quilting.	Vector Mode: Linear and Circular Interpolation Motion	VM, VP, CR, VS,VR, VA, VD, VE
Third axis must remain tangent to 2-D motion path, such as knife cutting.	Coordinated motion with Tangent Motion:	VM, VP, CR, VS,VA,VD, TN, VE
Electronic gearing where slave axes are scaled to master axis which can move in both directions.	Electronic Gearing	GA, GD, _GP, GR, GM (if gantry)
Master/slave where slave axes must follow a master such as conveyer speed.	Electronic Gearingwith Ramped Gearing	GA, GD, _GP, GR

Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Contour Mode	CM, CD, DT
Teaching or Record and Play Back	Contour Mode with Teach (Record and Play-Back)	CM, CD, DT, RA, RD, RC
Backlash Correction	Dual Loop (Auxiliary Encoder)	DV
Following a trajectory based on a master encoder position	Electronic Cam	EA, EM, EP, ET, EB, EG, EQ
Smooth motion while operating in independent axis positioning	Independent Motion Smoothing	IT
Smooth motion while operating in vector or linear interpolation positioning	Motion Smoothing	IT
Smooth motion while operating with stepper motors	Using the KS Command (Step Motor Smoothing):	KS
Gantry - two axes are coupled by gantry	Example - Gantry Mode	GR, GM

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the DMC-42x0 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-42x0 profiler. Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

The Begin (BG) command can be issued for all axes either simultaneously or independently. XYZ or W axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a begin. Note: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

Command Summary - Independent Axis

COMMAND	DESCRIPTION
PR x,y,z,w	Specifies relative distance
PA x,y,z,w	Specifies absolute position
SP x,y,z,w	Specifies slew speed
AC x,y,z,w	Specifies acceleration rate
DC x,y,z,w	Specifies deceleration rate
BG XYZW	Starts motion
ST XYZW	Stops motion before end of move
IP x,y,z,w	Changes position target
IT x,y,z,w	Time constant for independent motion smoothing
AM XYZW	Trippoint for profiler complete
MC XYZW	Trippoint for “in position”

The lower case specifiers (x,y,z,w) represent position values for each axis.

The DMC-42x0 also allows use of single axis specifiers such as PRY=2000

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Returns the speed for the axis specified by 'x'
_PAx	Returns current destination if 'x' axis is moving, otherwise returns the current commanded position if in a move.
_PRx	Returns current incremental distance specified for the 'x' axis

Example - Absolute Position Movement

```
PA 10000,20000          Specify absolute X,Y position
AC 1000000,1000000      Acceleration for X,Y
DC 1000000,1000000      Deceleration for X,Y
SP 50000,30000          Speeds for X,Y
BG XY                   Begin motion
```

Example - Multiple Move Sequence

Required Motion Profiles:

X-Axis	500 counts	Position
	20000 count/sec	Speed
	500000 counts/sec^2	Acceleration
Y-Axis	1000 counts	Position
	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration
Z-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec	Acceleration

This example will specify a relative position movement on X, Y and Z axes. The movement on each axis will be separated by 20 msec. Figure 6.1 shows the velocity profiles for the X,Y and Z axis.

```
#A                          Begin Program
PR 2000,500,100             Specify relative position movement of 2000, 500 and 100 counts for X,Y and Z axes.
SP 20000,10000,5000         Specify speed of 20000, 10000, and 5000 counts / sec
AC 500000,500000,500000     Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000,500000     Specify deceleration of 500000 counts / sec^2 for all axes
BG X                        Begin motion on the X axis
WT 20                       Wait 20 msec
BG Y                        Begin motion on the Y axis
WT 20                       Wait 20 msec
BG Z                        Begin motion on Z axis
EN                          End Program
```

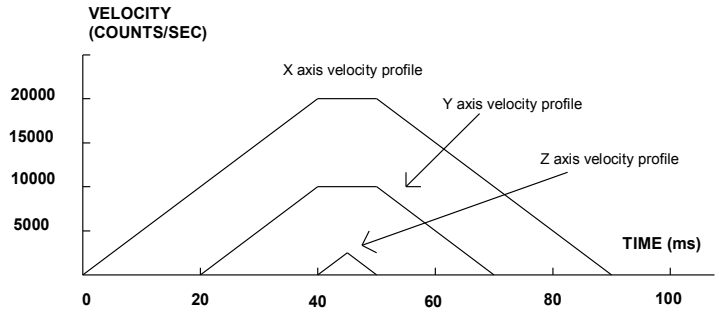


Figure 6.1: Velocity Profiles of XYZ

Notes on Figure 6.1: The X and Y axis have a 'trapezoidal' velocity profile, while the Z axis has a 'triangular' velocity profile. The X and Y axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The Z axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, hence, the slope of the rising and falling edges of all 3 velocity profiles are the same.

Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make a accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-42x0 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC x,y,z,w	Specifies acceleration rate
BG XYZW	Begins motion
DC x,y,z,w	Specifies deceleration rate
IP x,y,z,w	Increments position instantly
IT x,y,z,w	Time constant for independent motion smoothing
JG ±x,y,z,w	Specifies jog speed and direction
ST XYZW	Stops motion

Parameters can be set with individual axes specifiers such as JGY=2000 (set jog speed for Y axis to 2000).

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Returns the jog speed for the axis specified by 'x'
_TVx	Returns the actual velocity of the axis specified by 'x' (averaged over 0.25 sec)

Example - Jog in X only

Jog X motor at 50000 count/s. After X motor is at its jog speed, begin jogging Z in reverse direction at 25000 count/s.

```
#A
AC 20000,,20000        Specify X,Z acceleration of 20000 counts / sec
DC 20000,,20000        Specify X,Z deceleration of 20000 counts / sec
JG 50000,,-25000       Specify jog speed and direction for X and Z axis
BG X                   Begin X motion
AS X                   Wait until X is at speed
BG Z                   Begin Z motion
EN
```

Example - Joystick Jogging

The jog speed can also be changed using an analog input such as a joystick. Assume that for a 10 Volt input the speed must be 50000 counts/sec.

```
#JOY                   Label
JG0                    Set in Jog Mode
BGX                    Begin motion
#B                     Label for loop
V1 =@AN[1]             Read analog input
VEL=V1*50000/10        Compute speed
JG VEL                 Change JG speed
JP #B                  Loop
```

Position Tracking

The Galil controller may be placed in the position tracking mode to support changing the target of an absolute position move on the fly. New targets may be given in the same direction or the opposite direction of the current position target. The controller will then calculate a new trajectory based upon the new target and the acceleration, deceleration, and speed parameters that have been set. The motion profile in this mode is trapezoidal. There is not a set limit governing the rate at which the end point may be changed, however at the standard TM rate, the controller updates the position information at the rate of 1msec. The controller generates a profiled point every other sample, and linearly interpolates one sample between each profiled point. Some examples of applications that may use this mode are satellite tracking, missile tracking, random pattern polishing of mirrors or lenses, or any application that requires the ability to change the endpoint without completing the previous move.

The PA command is typically used to command an axis or multiple axes to a specific absolute position. For some applications such as tracking an object, the controller must proceed towards a target and have the ability to change the target during the move. In a tracking application, this could occur at any time during the move or at regularly scheduled intervals. For example if a robot was designed to follow a moving object at a specified distance and the path of the object wasn't known the robot would be required to constantly monitor the motion of the object that it was following. To remain within a specified distance it would also need to constantly update the position target it is moving towards. Galil motion controllers support this type of motion with the position tracking mode. This mode will allow scheduled or random updates to the current position target on the fly. Based on the new target the controller will either continue in the direction it is heading, change the direction it is moving, or decelerate to a stop.

The position tracking mode shouldn't be confused with the contour mode. The contour mode allows the user to generate custom profiles by updating the reference position at a specific time rate. In this mode, the position can be updated randomly or at a fixed time rate, but the velocity profile will always be trapezoidal with the parameters specified by AC, DC, and SP. Updating the position target at a specific rate will not allow the user to create a custom profile.

The following example will demonstrate the possible different motions that may be commanded by the controller in the position tracking mode. In this example, there is a host program that will generate the absolute position targets. The absolute target is determined based on the current information the host program has gathered on the object that it is tracking. The position tracking mode does allow for all of the axes on the controller to be in this mode, but for the sake of discussion, it is assumed that the robot is tracking only in the X dimension.

The controller must be placed in the position tracking mode to allow on the fly absolute position changes. This is performed with the PT command. To place the X axis in this mode, the host would issue PT1 to the controller if both X and Y axes were desired the command would be PT 1,1. The next step is to begin issuing PA command to the controller. The BG command isn't required in this mode, the SP, AC, and DC commands determine the shape of the trapezoidal velocity profile that the controller will use.

Example - Motion 1:

The host program determines that the first target for the controller to move to is located at 5000 encoder counts. The acceleration and deceleration should be set to 150,000 counts/sec2 and the velocity is set to 50,000 counts/sec. The command sequence to perform this is listed below.

```
#EX1
PT 1;'          Place the X axis in Position tracking mode
AC 150000;'     Set the X axis acceleration to 150000 counts/sec2
DC 150000;'     Set the X axis deceleration to 150000 counts/sec2
SP 50000;'      Set the X axis speed to 50000 counts/sec
PA 5000;'       Command the X axis to absolute position 5000 encoder counts
EN
```

The output from this code can be seen in Figure 6.2, a screen capture from the GalilTools scope.

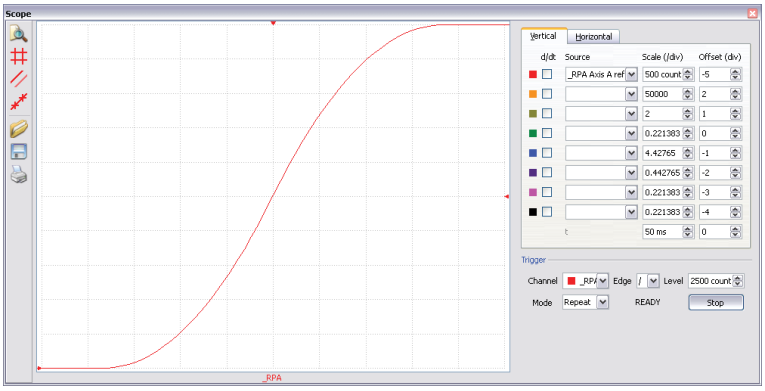


Figure 6.2: Position vs Time (msec) - Motion 1

Example - Motion 2:

The previous step showed the plot if the motion continued all the way to 5000, however partway through the motion, the object that was being tracked changed direction, so the host program determined that the actual target position should be 2000 counts at that time. Figure 6.2 shows what the position profile would look like if the move was allowed to complete to 5000 counts. The position was modified when the robot was at a position of 4200 counts (Figure 6.3). Note that the robot actually travels to a distance of almost 5000 counts before it turns around. This is a function of the deceleration rate set by the DC command. When a direction change is

commanded, the controller decelerates at the rate specified by the DC command. The controller then ramps the velocity in up to the value set with SP in the opposite direction traveling to the new specified absolute position. Figure 6.3 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 counts/sec before it is commanded to change direction.

The below code is used to simulate this scenario:

```
#EX2
PT 1;'  Place the X axis in Position tracking mode
AC 150000;'    Set the X axis acceleration to 150000 counts/sec2
DC 150000;'    Set the X axis deceleration to 150000 counts/sec2
SP 50000;'     Set the X axis speed to 50000 counts/sec
PA 5000;'      Command the X axis to abs position 5000 encoder counts
MF 4200
PA 2000;'      Change end point position to position 2000
EN
```

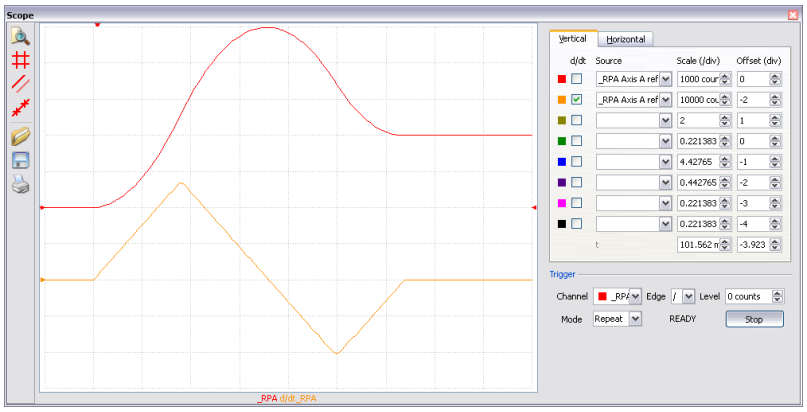


Figure 6.3: Position and Velocity vs Time (msec) for Motion 2

Example - Motion 3:

In this motion, the host program commands the controller to begin motion towards position 5000, changes the target to -2000, and then changes it again to 8000. Figure 6.4 shows the plot of position vs. time and velocity vs. time. Below is the code that is used to simulate this scenario:

```
#EX3
PT 1;' Place the X axis in Position tracking mode
AC 150000;' Set the X axis acceleration to 150000 counts/sec2
DC 150000;' Set the X axis deceleration to 150000 counts/sec2
SP 50000;' Set the X axis speed to 50000 counts/sec
PA 5000;' Command the X axis to abs position 5000 encoder counts
WT 300
PA -2000;' Change end point position to -2000
WT 200
PA 8000;' Change end point position to 8000
EN
```

Figure 6.5 demonstrates the use of motion smoothing (IT) on the velocity profile in this mode. The jerk in the system is also affected by the values set for AC and DC.

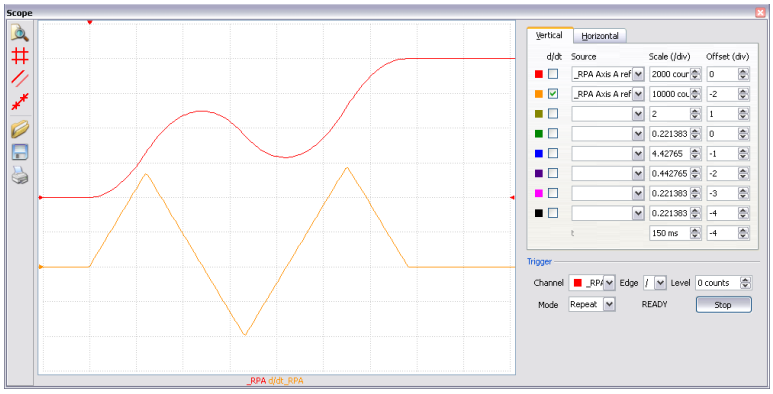


Figure 6.4: Position and Velocity vs Time (msec) for Motion 3

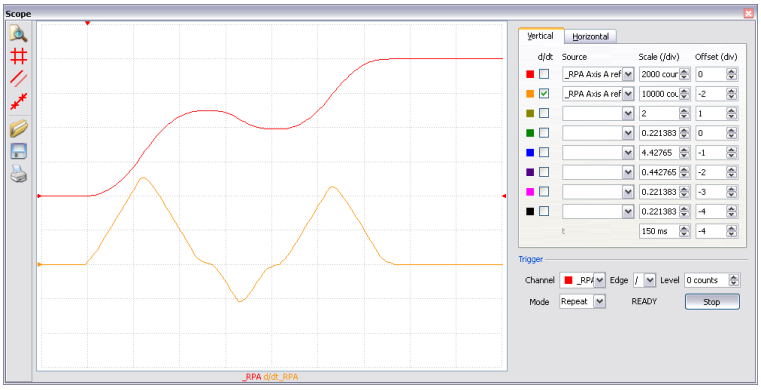


Figure 6.5: Position and Velocity vs Time (msec) for Motion 3 with IT 0.1

Note the controller treats the point where the velocity passes through zero as the end of one move, and the beginning of another move. IT is allowed, however it will introduce some time delay.

Trippoints

Most trippoints are valid for use while in the position tracking mode. There are a few exceptions to this; the AM and MC commands may not be used while in this mode. It is recommended that MF, MR, or AP be used, as they involve motion in a specified direction, or the passing of a specific absolute position.

Command Summary – Position Tracking Mode

COMMAND	DESCRIPTION
AC n,n,n,n,n,n,n,n	Acceleration settings for the specified axes
AP n,n,n,n,n,n,n,n	Trippoint that holds up program execution until an absolute position has been reached
DC n,n,n,n,n,n,n,n	Deceleration settings for the specified axes
MF n,n,n,n,n,n,n,n	Trippoint to hold up program execution until n number of counts have passed in the forward direction. Only one axis at a time may be specified.
MR n,n,n,n,n,n,n,n	Trippoint to hold up program execution until n number of counts have passed in the reverse direction. Only one axis at a time may be specified.
PT n,n,n,n,n,n,n,n	Command used to enter and exit the Trajectory Modification Mode
PA n,n,n,n,n,n,n,n	Command Used to specify the absolute position target
SP n,n,n,n,n,n,n,n	Speed settings for the specified axes

Linear Interpolation Mode

The DMC-42x0 provides a linear interpolation mode for 2 or more axes. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM command selects the Linear Interpolation mode and axes for interpolation. For example, LM YZ selects only the Y and Z axes for linear interpolation.

When using the linear interpolation mode, the LM command only needs to be specified once unless the axes for linear interpolation change.

Specifying Linear Segments

The command LI x,y,z,w or LI a,b,c,d,e,f,g,h specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The clear sequence (CS) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the DMC-42x0 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent at PC bus speeds.

The instruction _CS returns the segment counter. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being processed.

Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The DMC-42x0 computes the vector speed based on the axes specified in the LM mode. For example, LM XYZ designates linear interpolation for the X,Y and Z axes. The vector speed for this example would be computed using the equation:

VS2=XS2+YS2+ZS2, where XS, YS and ZS are the speed of the X,Y and Z axes.

The controller always uses the axis specifications from LM, not LI, to compute the speed.

IT is used to set the S-curve smoothing constant for coordinated moves. The command AV n is the ‘After Vector’ trippoint, which halts program execution until the vector distance of n has been reached.

An Example of Linear Interpolation Motion:

```
#LMOVE          label
DP 0,0          Define position of X and Y axes to be 0
LMXY            Define linear mode between X and Y axes.
LI 5000,0       Specify first linear segment
LI 0,5000       Specify second linear segment
LE              End linear segments
VS 4000         Specify vector speed
BGS             Begin motion sequence
AV 4000         Set trippoint to wait until vector distance of 4000 is reached
VS 1000         Change vector speed
AV 5000         Set trippoint to wait until vector distance of 5000 is reached
VS 4000         Change vector speed
EN              Program end
```

In this example, the XY system is required to perform a 90° turn. In order to slow the speed around the corner, we use the AV 4000 trippoint, which slows the speed to 1000 count/s. Once the motors reach the corner, the speed is increased back to 4000 counts / s.

Specifying Vector Speed for Each Segment

The instruction VS has an immediate effect and, therefore, must be given at the required time. In some applications, such as CNC, it is necessary to attach various speeds to different motion segments. This can be done by two functions: < n and > m

For example: LI x,y,z,w < n >m

The first command, < n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

As an example, consider the following program.

```
#ALT                            Label for alternative program
DP 0,0                          Define Position of X and Y axis to be 0
LMXY                            Define linear mode between X and Y axes.
LI 4000,0 <4000 >1000           Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000,1000 < 4000 >1000       Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0,5000 < 4000 >1000          Specify third linear segment with a vector speed of 4000 and end speed 1000
LE                              End linear segments
BGS                             Begin motion sequence
EN                              Program end
```

Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

Command Summary - Linear Interpolation

COMMAND	DESCRIPTION
LM xyzw LM abcdefgh	Specify axes for linear interpolation (same) controllers with 5 or more axes
LM?	Returns number of available spaces for linear segments in DMC-42x0 sequence buffer. Zero means buffer full. 511 means buffer empty.
LI x,y,z,w < n LI a,b,c,d,e,f,g,h < n	Specify incremental distances relative to current position, and assign vector speed n.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BGS	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
LE?	Returns the length of the vector (resets after 2147483647)
AMS	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n
IT	S curve smoothing constant for vector moves

Operand Summary - Linear Interpolation

OPERAND	DESCRIPTION
_AV	Return distance traveled
_CS	Segment counter - returns number of the segment in the sequence, starting at zero.
_LE	Returns length of vector (resets after 2147483647)
_LM	Returns number of available spaces for linear segments in DMC-42x0 sequence buffer. Zero means buffer full. 511 means buffer empty.
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m=X,Y,Z or W or A,B,C,D,E,F,G or H)

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the X axis moves toward the point X=5000. Suppose that when X=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS, _VPX and _VPY will be zero.

Now suppose that the interrogation is repeated at the second segment when Y=2000. The value of _AV at this point is 7000, _CS equals 1, _VPX=5000 and _VPY=0.

Example - Linear Move

Make a coordinated linear move in the ZW plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec2.

```
LM ZW              Specify axes for linear interpolation
LI,,40000,30000    Specify ZW distances
LE                 Specify end move
VS 100000          Specify vector speed
VA 1000000         Specify vector acceleration
VD 1000000         Specify vector deceleration
BGS                Begin sequence
```

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VZ and VW. The axis speeds are determined by the controller from:

$$VS = \sqrt{VZ^2 + VW^2}$$

The result is shown in Figure 6.6: Linear Interpolation.

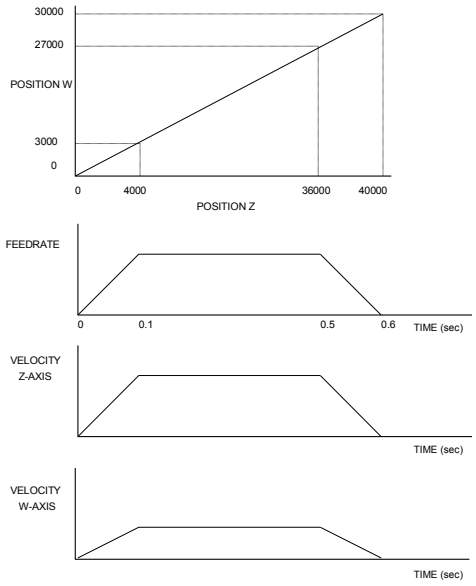


Figure 6.6: Linear Interpolation

Example - Multiple Moves

This example makes a coordinated linear move in the XY plane. The Arrays VX and VY are used to store 750 incremental distances which are filled by the program #LOAD.

`#LOAD`	Load Program
`DM VX [750],VY [750]`	Define Array
`COUNT=0`	Initialize Counter
`N=0`	Initialize position increment
`#LOOP`	LOOP
`VX [COUNT]=N`	Fill Array VX
`VY [COUNT]=N`	Fill Array VY
`N=N+10`	Increment position
`COUNT=COUNT+1`	Increment counter
`JP #LOOP,COUNT<750`	Loop if array not full
`#A`	Label
`LM XY`	Specify linear mode for XY
`COUNT=0`	Initialize array counter
`#LOOP2;JP#LOOP2,_LM=0`	If sequence buffer full, wait
`JS#C,COUNT=500`	Begin motion on 500th segment
`LI VX[COUNT],VY[COUNT]`	Specify linear segment
`COUNT=COUNT+1`	Increment array counter
`JP #LOOP2,COUNT<750`	Repeat until array done
`LE`	End Linear Move
`AMS`	After Move sequence done
`MG "DONE"`	Send Message
`EN`	End program
`#C;BGS;EN`	Begin Motion Subroutine

Vector Mode: Linear and Circular Interpolation Motion

The DMC-42x0 allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the prescribed vector speed even at transitions between linear and circular segments. The

DMC-42x0 performs all the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

The coordinated motion mode is similar to the linear interpolation mode. Any pair of two axes may be selected for coordinated motion consisting of linear and circular segments. In addition, a third axis can be controlled such that it remains tangent to the motion of the selected pair of axes. Note that only one pair of axes can be specified for coordinated motion at any given time.

The command VM m,n,p where 'm' and 'n' are the coordinated pair and p is the tangent axis (Note: the commas which separate m,n and p are not necessary). For example, VM XWZ selects the XW axes for coordinated motion and the Z-axis as the tangent.

Specifying the Coordinate Plane

The DMC-42x0 allows for 2 separate sets of coordinate axes for linear interpolation mode or vector mode. These two sets are identified by the letters S and T.

To specify vector commands the coordinate plane must first be identified. This is done by issuing the command CAS to identify the S plane or CAT to identify the T plane. All vector commands will be applied to the active coordinate system until changed with the CA command.

Specifying Vector Segments

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note: This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command, VP x,y specifies the coordinates of the end points of the vector movement with respect to the starting point. Non-sequential axis do not require comma delimitation. The command, CR r,q,d define a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (CCW) rotation is positive.

Up to 511 segments of CR or VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command requires the controller to decelerate to a stop following the last motion requirement. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

It is the responsibility of the user to keep enough motion segments in the DMC-42x0 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

The operand _CS can be used to determine the value of the segment counter.

Additional commands

The commands VS n, VA n and VD n are used for specifying the vector speed, acceleration, and deceleration.

IT is the s curve smoothing constant used with coordinated motion.

Specifying Vector Speed for Each Segment:

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x,y < n >m

CR r, ɵ,δ< n >m

The first command, <n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR 0.5 results in the specification VS 2000 to be divided by two.

Compensating for Differences in Encoder Resolution:

By default, the DMC-42x0 uses a scale factor of 1:1 for the encoder resolution when used in vector mode. If this is not the case, the command, ES can be used to scale the encoder counts. The ES command accepts two arguments which represent the number of counts for the two encoders used for vector motion. The smaller ratio of the two numbers will be multiplied by the higher resolution encoder. For more information, see ES command in the Command Reference.

Trippoints:

The AV n command is the After Vector trippoint, which waits for the vector relative distance of n to occur before executing the next command in a program.

Tangent Motion:

Several applications, such as cutting, require a third axis (i.e. a knife blade), to remain tangent to the coordinated motion path. To handle these applications, the DMC-42x0 allows one axis to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

`VM m,n,p`	m,n specifies coordinated axes p specifies tangent axis such as X,Y,Z,W p=N turns off tangent axis

Before the tangent mode can operate, it is necessary to assign an axis via the VM command and define its offset and scale factor via the TN m,n command. m defines the scale factor in counts/degree and n defines the tangent position that equals zero degrees in the coordinated motion plane. The operand _TN can be used to return the initial position of the tangent axis.

Example:

Assume an XY table with the Z-axis controlling a knife. The Z-axis has a 2000 quad counts/rev encoder and has been initialized after power-up to point the knife in the +Y direction. A 180° circular cut is desired, with a radius of 3000, center at the origin and a starting point at (3000,0). The motion is CCW, ending at (-3000,0). Note that the 0° position in the XY plane is in the +X direction. This corresponds to the position -500 in the Z-axis, and defines the offset. The motion has two parts. First, X,Y and Z are driven to the starting point, and later, the cut is performed. Assume that the knife is engaged with output bit 0.

```
#EXAMPLE              Example program
VM XYZ                XY coordinate with Z as tangent
TN 2000/360,-500      2000/360 counts/degree, position -500 is 0 degrees in XY plane
CR 3000,0,180         3000 count radius, start at 0 and go to 180 CCW
VE                    End vector
CB0                   Disengage knife
PA 3000,0,_TN         Move X and Y to starting position, move Z to initial tangent position
BG XYZ                Start the move to get into position
AM XYZ                When the move is complete
SB0                   Engage knife
WT50                  Wait 50 msec for the knife to engage
BGS                   Do the circular cut
AMS                   After the coordinated move is complete
CB0                   Disengage knife
MG "ALL DONE"
EN                    End program
```

Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION
VM m,n	Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis.
VP m,n	Return coordinate of last point, where m=X,Y,Z or W.
CR r,ɵ,δ<n>m	Specifies arc segment where r is the radius, Θ is the starting angle and ΔΘ is the travel angle. Positive direction is CCW.
VS s,t	Specify vector speed or feed rate of sequence.
VA s,t	Specify vector acceleration along the sequence.
VD s,t	Specify vector deceleration along the sequence.
VR s,t	Specify vector speed ratio
BGST	Begin motion sequence, S or T
CSST	Clear sequence, S or T
AV s,t	Trippoint for After Relative Vector distance.
AMST	Holds execution of next command until Motion Sequence is complete.
TN m,n	Tangent scale and offset.
ES m,n	Ellipse scale factor.
IT s,t	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in DMC-42x0 sequence buffer. Zero means buffer is full. 511 means buffer is empty.
CAS or CAT	Specifies which coordinate system is to be active (S or T)

Operand Summary - Coordinated Motion Sequence

OPERAND	DESCRIPTION
_VPM	The absolute coordinate of the axes at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in DMC-42x0 sequence buffer. Zero means buffer is full. 511 means buffer is empty.
_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operands _VPX and _VPY can be used to return the coordinates of the last point specified along the path.

Example:

Traverse the path shown in Figure 6.7. Feed rate is 20000 counts/sec. Plane of motion is XY

```
VM XY                 Specify motion plane
VS 20000              Specify vector speed
VA 1000000            Specify vector acceleration
VD 1000000            Specify vector deceleration
VP -4000,0            Segment AB
CR 1500,270,-180      Segment BC
VP 0,3000             Segment CD
CR 1500,90,-180       Segment DA
VE                    End of sequence
BGS                   Begin Sequence
```

The resulting motion starts at the point A and moves toward points B, C, D, A. Suppose that we interrogate the controller when the motion is halfway between the points A and B.

The value of _AV is 2000

The value of _CS is 0

_VPX and _VPY contain the absolute coordinate of the point A

Suppose that the interrogation is repeated at a point, halfway between the points C and D.

The value of _AV is 4000+1500π+2000=10,712

The value of _CS is 2

_VPX,_VPY contain the coordinates of the point C

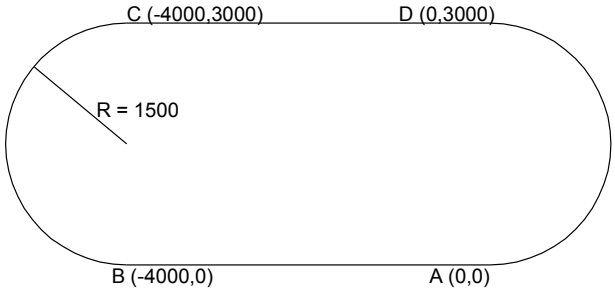


Figure 6.7: The Required Path

Vector Mode - Mathematical Analysis

The terms of coordinated motion are best explained in terms of the vector motion. The vector velocity, Vs, which is also known as the feed rate, is the vector sum of the velocities along the X and Y axes, Vx and Vy.

$$Vs = \sqrt{Vx^2 + Vy^2}$$

The vector distance is the integral of Vs, or the total distance traveled along the path. To illustrate this further, suppose that a string was placed along the path in the X-Y plane. The length of that string represents the distance traveled by the vector motion.

The vector velocity is specified independently of the path to allow continuous motion. The path is specified as a collection of segments. For the purpose of specifying the path, define a special X-Y coordinate system whose origin is the starting point of the sequence. Each linear segment is specified by the X-Y coordinate of the final point

expressed in units of resolution, and each circular arc is defined by the arc radius, the starting angle, and the angular width of the arc. The zero angle corresponds to the positive direction of the X-axis and the CCW direction of rotation is positive. Angles are expressed in degrees, and the resolution is 1/256th of a degree. For example, the path shown in Figure A.8 is specified by the instructions:

```
VP 0,10000
CR 10000, 180, -90
VP 20000, 20000
```

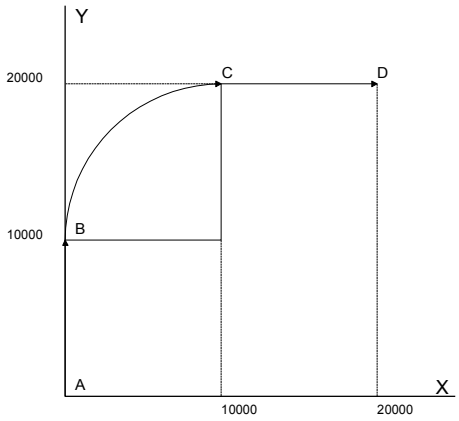


Figure A.8: X-Y Motion Path

The first line describes the straight line vector segment between points A and B. The next segment is a circular arc, which starts at an angle of 180° and traverses -90°. Finally, the third line describes the linear segment between points C and D. Note that the total length of the motion consists of the segments:

A-B	Linear	10000 units
B-C	Circular	$\frac{R\lvert\Delta\theta\rvert 2\pi}{360}$ = 15708
C-D	Linear	10000
	Total	35708 counts

In general, the length of each linear segment is

$$L_k = \sqrt{Xk^2 + Yk^2}$$

Where Xk and Yk are the changes in X and Y positions along the linear segment. The length of the circular arc is

$$L_k = R_k\lvert\Delta\Theta_k\rvert 2\pi/360$$

The total travel distance is given by

$$D = \sum_{k=1}^{n} L_k$$

The velocity profile may be specified independently in terms of the vector velocity and acceleration.

For example, the velocity profile corresponding to the path of Figure A.8 may be specified in terms of the vector speed and acceleration.

```
VS 100000
VA 2000000
```

The resulting vector velocity is shown in Figure A.9.

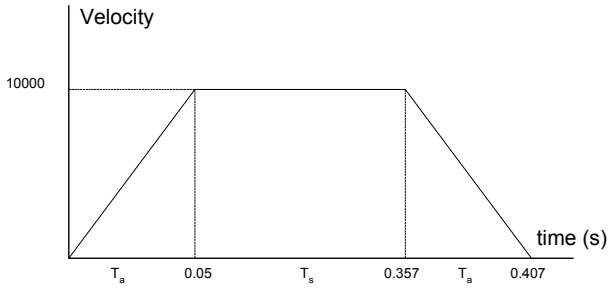


Figure A.9: Vector Velocity Profile

The acceleration time, Ta, is given by

$$T_a = \frac{VS}{VA} = \frac{100000}{2000000} = 0.05s$$

The slew time, Ts, is given by

$$T_s = \frac{D}{VS} - T_a = \frac{35708}{100000} - 0.05 = 0.307s$$

The total motion time, Tt, is given by:

$$T_t = \frac{D}{VS} + T_a = 0.407s$$

The velocities along the X and Y axes are such that the direction of motion follows the specified path, yet the vector velocity fits the vector speed and acceleration requirements.

For example, the velocities along the X and Y axes for the path shown in Figure A.8 are given in Figure A.10.

Figure A.10 shows the vector velocity. It also indicates the position point along the path starting at A and ending at D. Between the points A and B, the motion is along the Y axis. Therefore,

Vy = Vs

and

Vx = 0

Between the points B and C, the velocities vary gradually and finally, between the points C and D, the motion is in the X direction.

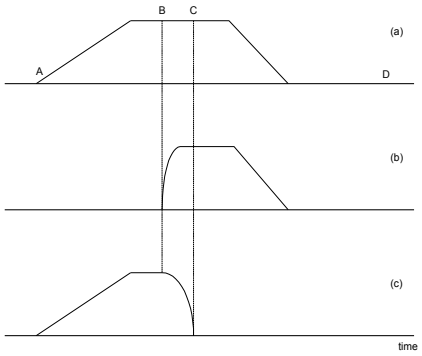


Figure A.10: Vector Axes Velocities

Electronic Gearing

This mode allows up to 8 axes to be electronically geared to some master axes. The masters may rotate in both directions and the geared axes will follow at the specified gear ratio. The gear ratio may be different for each axis and changed during motion.

The command GAX yzw or GA ABCDEFGH specifies the master axes. GR x,y,z,w specifies the gear ratios for the slaves where the ratio may be a number between ±127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode (enabled with the command GM) allows the gearing to stay enabled even if a limit is hit or an ST command is issued. GR 0,0,0,0 turns off gearing in both modes.

The command GM x,y,z,w select the axes to be controlled under the gantry mode. The parameter 1 enables gantry mode, and 0 disables it.

GR causes the specified axes to be geared to the actual position of the master. The master axis is commanded with motion commands such as PR, PA or JG.

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the commanded position master is by the letter, C. For example, GACX indicates that the gearing is the commanded position of X.

An alternative gearing method is to synchronize the slave motor to the commanded vector motion of several axes performed by GAS. For example, if the X and Y motor form a circular motion, the Z axis may move in proportion to the vector move. Similarly, if X,Y and Z perform a linear interpolation move, W can be geared to the vector move.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, or JG, commands, or VP, or LI.

Ramped Gearing

In some applications, especially when the master is traveling at high speeds, it is desirable to have the gear ratio ramp gradually to minimize large changes in velocity on the slave axis when the gearing is engaged. For example if the master axis is already traveling at 500,000 counts/sec and the slave will be geared at a ratio of 1:1 when the gearing is engaged, the slave will instantly develop following error, and command maximum current to the motor. This can be a large shock to the system. For many applications it is acceptable to slowly ramp the engagement of gearing over a greater time frame. Galil allows the user to specify an interval of the master axis over which the gearing will be engaged. For example, the same master X axis in this case travels at 500,000 counts/sec, and the gear ratio is 1:1, but the gearing is slowly engaged over 30,000 counts of the master axis, greatly diminishing the initial shock to the slave axis. Figure 6.12 below shows the velocity vs. time profile for instantaneous gearing. Figure 6.14 shows the velocity vs. time profile for the gradual gearing engagement.

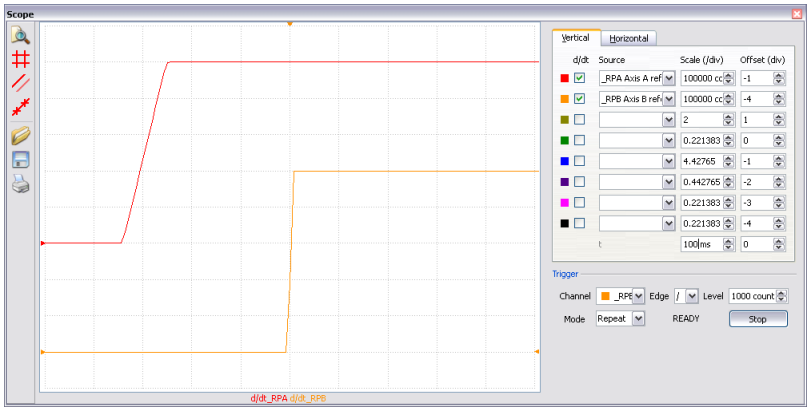


Figure 6.11: Velocity counts/sec vs. Time (msec) Instantaneous Gearing Engagement

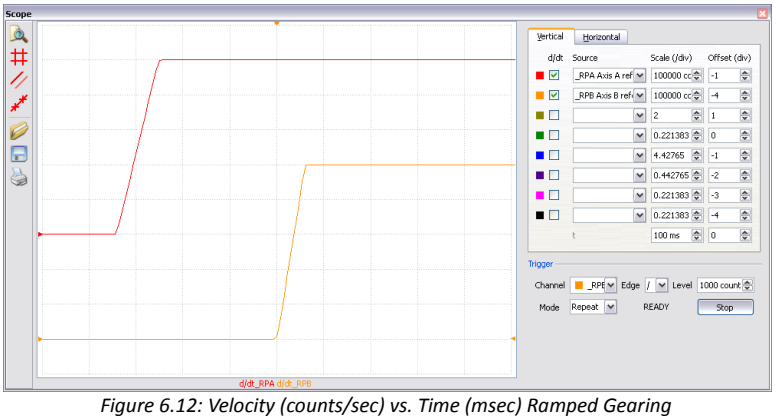


Figure 6.12: Velocity (counts/sec) vs. Time (msec) Ramped Gearing

The slave axis for each figure is shown on the bottom portion of the figure; the master axis is shown on the top portion. The shock to the slave axis will be significantly less in Figure 6.14 than in Figure 6.12. The ramped gearing does have one consequence. There isn't a true synchronization of the two axes, until the gearing ramp is complete. The slave will lag behind the true ratio during the ramp period. If exact position synchronization is required from the point gearing is initiated, then the position must be commanded in addition to the gearing. The controller

keeps track of this position phase lag with the _GP operand. The following example will demonstrate how the command is used.

Example – Electronic Gearing Over a Specified Interval

Objective Run two geared motors at speeds of 1.132 and -.045 times the speed of an external master. Because the master is traveling at high speeds, it is desirable for the speeds to change slowly.

Solution: Use a DMC-4230 controller where the Z-axis is the master and X and Y are the geared axes. We will implement the gearing change over 6000 counts (3 revolutions) of the master axis.

```
MO Z                    Turn Z off, for external master
GA Z, Z                 Specify Z as the master axis for both X and Y.
GD 6000,6000            Specify ramped gearing over 6000 counts of the master axis.
GR 1.132,-.045          Specify gear ratios
```

Question: What is the effect of the ramped gearing?

Answer: Below, in the example titled Electronic Gearing, gearing would take effect immediately. From the start of gearing if the master traveled 6000 counts, the slaves would travel 6792 counts and 270 counts.

Using the ramped gearing, the slave will engage gearing gradually. Since the gearing is engaged over the interval of 6000 counts of the master, the slave will only travel ~3396 counts and ~135 counts respectively. The difference between these two values is stored in the _GPn operand. If exact position synchronization is required, the IP command is used to adjust for the difference.

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies master axes for gearing where: n = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master
	n = CX,CY,CZ, CW **or** CA, CB,CC,CD,CE,CF,CG,CH for commanded position.
	n = DX,DY,DZ or DW or DA, DB, DC, DD, DE, DF,DG,DH for auxiliary encoders
	n = S or T for gearing to coordinated motion.
GD a,b,c,d,e,f,g,h	Sets the distance the master will travel for the gearing change to take full effect.
_GPn	This operand keeps track of the difference between the theoretical distance traveled if gearing changes took effect immediately, and the distance traveled since gearing changes take effect over a specified interval.
GR a,b,c,d,e,f,g,h	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GM a,b,c,d,e,f,g,h	X = 1 sets gantry mode, 0 disables gantry mode
MR x,y,z,w	Trippoint for reverse motion past specified value. Only one field may be used.
MF x,y,z,w	Trippoint for forward motion past specified value. Only one field may be used.

Example - Simple Master Slave

Master axis moves 10000 counts at slew speed of 100000 counts/sec. Y is defined as the master. X,Z,W are geared to master at ratios of 5,-.5 and 10 respectively.

```
GA Y,,Y,Y               Specify master axes as Y
GR 5,,-.5,10            Set gear ratios
PR ,10000               Specify Y position
SP ,100000              Specify Y speed
BGY                     Begin motion
```

Example - Electronic Gearing

Objective: Run two geared motors at speeds of 1.132 and -0.045 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder).

Solution: Use a DMC-4230 controller, where the Z-axis is the master and X and Y are the geared axes.

```
MO Z                Turn Z off, for external master
GA Z, Z             Specify Z as the master axis for both X and Y.
GR 1.132,-.045      Specify gear ratios
```

Now suppose the gear ratio of the X-axis is to change on-the-fly to 2. This can be achieved by commanding:

```
GR 2                Specify gear ratio for X axis to be 2
```

Example - Gantry Mode

In applications where both the master and the follower are controlled by the DMC-42x0 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

For example, assume that a gantry is driven by two axes, X,Y, on both sides. This requires the gantry mode for strong coupling between the motors. The X-axis is the master and the Y-axis is the follower. To synchronize Y with the commanded position of X, use the instructions:

```
GA, CX              Specify the commanded position of X as master for Y.
GR,1                Set gear ratio for Y as 1:1
GM,1                Set gantry mode
PR 3000             Command X motion
BG X                Start motion on X axis
```

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command

```
IP ,10              Specify an incremental position movement of 10 on Y axis.
```

Under these conditions, this IP command is equivalent to:

```
PR,10               Specify position relative movement of 10 on Y axis
BGY                 Begin motion on Y axis
```

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

Example - Synchronize two conveyor belts with trapezoidal velocity correction

```
GA,X                Define X as the master axis for Y.
GR,2                Set gear ratio 2:1 for Y
PR,300              Specify correction distance
SP,5000             Specify correction speed
AC,100000           Specify correction acceleration
DC,100000           Specify correction deceleration
BGY                 Start correction
```

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of several axes of motion. Up to 7 axes can be slaved to one master axis. The master axis encoder must be input through a main encoder port.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes. For example, the DMC-4280 controllers may have one master and up to seven slaves.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis Y, when the master is X. Such a graphic relationship is shown in Figure 6.13.

Step 1. Selecting the master axis

The first step in the electronic cam mode is to select the master axis. This is done with the instruction

EAp where p = X,Y,Z,W,E,F,G,H

p is the selected master axis

For the given example, since the master is x, we specify EAX

Step 2. Specify the master cycle and the change in the slave axis (or axes).

In the electronic cam mode, the position of the master is always expressed modulo one cycle. In this example, the position of x is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

```
EM x,y,z,w
```

where x,y,z,w specify the cycle of the master and the total change of the slaves over one cycle.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instruction:

```
EM 6000,1500
```

Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

```
EP m,n
```

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

```
EP 2000,0
```

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

```
ET[n]=x,y,z,w
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameters x,y,z,w indicate the corresponding slave position. For this example, the table may be specified by

```
ET[0]=,0
ET[1]=,3000
ET[2]=,2250
ET[3]=,1500
```

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

```
EB n
```

where n=1 enables ECAM mode and n=0 disables ECAM mode.

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

```
EG x,y,z,w
```

where x,y,z,w are the master positions at which the corresponding slaves must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

Step 7. Disengage the slave motion

To disengage the cam, use the command

```
EQ x,y,z,w
```

where x,y,z,w are the master positions at which the corresponding slave axes are disengaged.

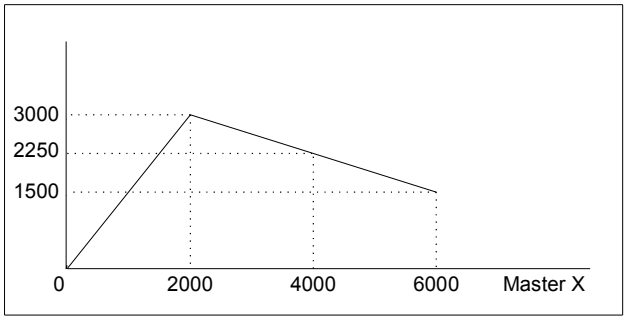


Figure 6.13: Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

To illustrate the complete process, consider the cam relationship described by

the equation:

$Y = 0.5 * X + 100 \sin(0.18 * X)$

where X is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAX defines X as the master axis. The cycle of the master is

2000. Over that cycle, Y varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18X and X varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of Y according to the equation and assigns the values to the table with the instruction ET[N] = ,Y.

INSTRUCTION	INTERPRETATION
`#SETUP`	Label
`EAX`	Select X as master
`EM 2000,1000`	Cam cycles
`EP 20,0`	Master position increments
`N = 0`	Index
`#LOOP`	Loop to construct table from equation
`P = N*3.6`	Note 3.6 = 0.18 * 20
`S = @SIN [P]*100`	Define sine position
`Y = N*10+S`	Define slave position
`ET [N] =, Y`	Define table
`N = N+1`	
`JP #LOOP, N<=100`	Repeat the process
`EN`	

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: X = 1000 and Y = 500. This implies that Y must be driven to that point to avoid a jump.

This is done with the program:

INSTRUCTION	INTERPRETATION
`#RUN`	Label
`EB1`	Enable cam
`PA,500`	starting position
`SP,5000`	Y speed
`BGY`	Move Y motor
`AM`	After Y moved
`AI1`	Wait for start signal
`EG,1000`	Engage slave
`AI - 1`	Wait for stop signal
`EQ,1000`	Disengage slave
`EN`	End

Command Summary - Electronic CAM

Command	Description
`EA p`	Specifies master axes for electronic cam where: p = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master or M or N a for virtual axis master
`EB n`	Enables the ECAM

EC n	ECAM counter - sets the index into the ECAM table
EG x,y,z,w	Engages ECAM
EM x,y,z,w	Specifies the change in position for each axis of the CAM cycle
EP m,n	Defines CAM table entry size and offset
EQ m,n	Disengages ECAM at specified position
ET[n]	Defines the ECAM table entries
EW	Widen Segment (see Application Note #2444)
EY	Set ECAM cycle count

Operand Summary - Electronic CAM

Command	Description
_EB	Contains State of ECAM
_EC	Contains current ECAM index
_EGx	Contains ECAM status for each axis
_EM	Contains size of cycle for each axis
_EP	Contains value of the ECAM table interval
_EQx	Contains ECAM status for each axis
_EY	Set ECAM cycle count

Example - Electronic CAM

The following example illustrates a cam program with a master axis, Z, and two slaves, X and Y.

INSTRUCTION	INTERPRETATION
#A;V1=0	Label; Initialize variable
PA 0,0;BGXY;AMXY	Go to position 0,0 on X and Y axes
EA Z	Z axis as the Master for ECAM
EM 0,0,4000	Change for Z is 4000, zero for X, Y
EP400,0	ECAM interval is 400 counts with zero start
ET[0]=0,0	When master is at 0 position; 1st point.
ET[1]=40,20	2nd point in the ECAM table
ET[2]=120,60	3rd point in the ECAM table
ET[3]=240,120	4th point in the ECAM table
ET[4]=280,140	5th point in the ECAM table
ET[5]=280,140	6th point in the ECAM table
ET[6]=280,140	7th point in the ECAM table
ET[7]=240,120	8th point in the ECAM table
ET[8]=120,60	9th point in the ECAM table
ET[9]=40,20	10th point in the ECAM table
ET[10]=0,0	Starting point for next cycle
EB 1	Enable ECAM mode
JGZ=4000	Set Z to jog at 4000
EG 0,0	Engage both X and Y when Master = 0
BGZ	Begin jog on Z axis
#LOOP;JP#LOOP,V1=0	Loop until the variable is set
EQ2000,2000	Disengage X and Y when Master = 2000
MF,, 2000	Wait until the Master goes to 2000
ST Z	Stop the Z axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller. Figure 6.14 shows the GalilTools scope capture of the ECAM profile. This shows how the motion will be seen during the ECAM cycles. The first trace is for the A axis, the second trace shows the cycle on the B axis and the third trace shows the cycle of the C axis.

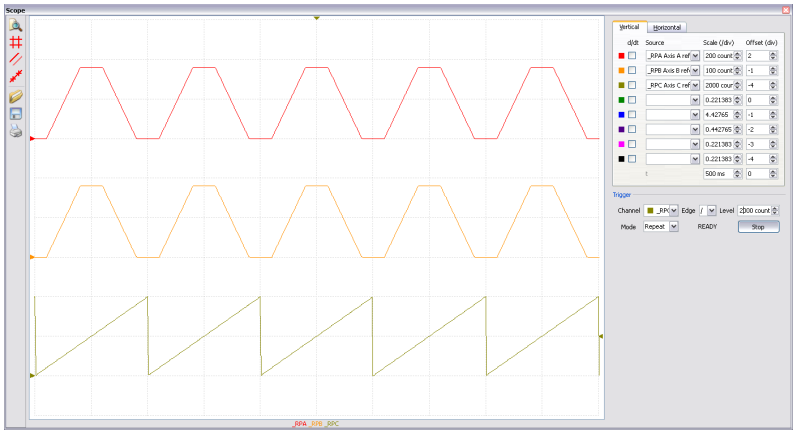


Figure 6.14: ECAM cycle with Z axis as master

PVT Mode

The DMC-42x0 controllers now supports a mode of motion referred to as “PVT.” This mode allows arbitrary motion profiles to be defined by position, velocity and time individually on all 8 axes. This motion is designed for systems where the load must traverse a series of coordinates with no discontinuities in velocity. By specifying the target position, velocity and time to achieve those parameters the user has control over the velocity profile. Taking advantage of the built in buffering the user can create virtually any profile including those with infinite path lengths.

Specifying PVT Segments

PVT segments must be entered one axis at a time using the PVn command. The PV command includes the target distance to be moved and target velocity to be obtained over the specified timeframe. Positions are entered as relative moves, similar to the standard PR command, in units of encoder counts and velocity is entered in counts/second. The controller will interpolate the motion profile between subsequent PV commands using a 3rd order polynomial equation. During a PV segment, jerk is held constant, and accelerations, velocities, and positions will be calculated every other sample.

Motion will not begin until a BT command is issued, much like the standard BG command. This means that the user can fill the PVT buffer for each axis prior to motion beginning. The BT command will ensure that all axes begin motion simultaneously. It is not required for the “t” value for each axis to be the same, however if they are then the axes will remain coordinated. Each axis has a 255 segment buffer. This buffer is a FIFO and the available space can be queried with the operand _PVn. As the buffer empties the user can add more PVT segments.

Exiting PVT Mode

To exit PVT mode the user must send the segment command PVn=0,0,0. This will exit the mode once the segment is reached in the buffer. To avoid an abrupt stop the user should slow the motion to a zero velocity prior to executing this command. The controller will instantly command a zero velocity once a PVn=0,0,0 is executed. In addition, a ST command will also exit PVT mode. Motion will come to a controlled stop using the DC value for deceleration. The same controlled stop will occur if a limit switch is activated in the direction of motion. As a result, the controller will be switched to a jog mode of motion.

Error Conditions and Stop Codes

If the buffer is allowed to empty while in PVT mode then the profiling will be aborted and the motor will come to a controlled stop on that axis with a deceleration specified by the DC command. Also, PVT mode will be exited and the stop code will be set to 32. During normal operation of PVT mode the stop code will be 30. If PVT mode is exited normally (PVn=0,0,0), then the stop code will be set to 31.

Additional PVT Information

It is the users’ responsibility to enter PVT data that the system’s mechanics and power system can respond to in a reasonable manner. Because this mode of motion is not constrained by the AC, DC or SP values, if a large velocity or position is entered with a short period to achieve it, the acceleration can be very high, beyond the capabilities of the system, resulting in excessive position error. The position and velocity at the end of the segment are guaranteed to be accurate but it is important to remember that the required path to obtain the position and velocity in the specified time may be different based on the PVT values. Mismatched values for PVT can result in different interpolated profiles than expected but the final velocity and position will be accurate.

The “t” value is entered in samples, which will depend on the TM setting. With the default TM of 1000, one sample is 976us. This means that a “t” value of 1024 will yield one second of motion. The velocity value, “v” will always be in units of counts per second, regardless of the TM setting. PVT mode is not available in the “-FAST” version of the firmware. If this is required please consult Galil.

Command Summary – PVT

COMMAND	DESCRIPTION
PVa = p,v,t	Specifies the segment of axis 'a' for a incremental PVT segment of 'p' counts, an end speed of 'v' counts/sec in a total time of 't' samples.
_PVa	Contains the number of PV segments available in the PV buffer for a specified axes.
BT	Begin PVT mode
_BTa	Contains the number PV segments that have executed

PVT Examples

Parabolic Velocity Profile

In this example we will assume that the user wants to start from zero velocity, accelerate to a maximum velocity of 1000 counts/second in 1 second and then back down to 0 counts/second within an additional second. The velocity profile would be described by the following equation and shown in Figure 6.15.

$$v(t) = -1000(t-1)^2 + 1000$$

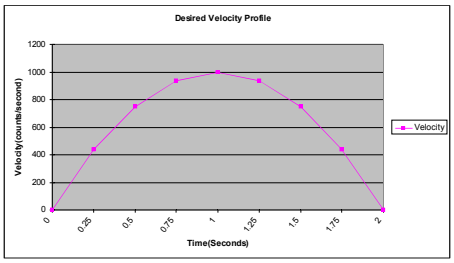


Figure 6.15: Parabolic Velocity Profile

To accomplish this we need to calculate the desired velocities and change in positions. In this example we will assume a delta time of ¼ of a second, which is 256 samples (1024 samples = 1 second with the default TM of 1000).

Velocity(counts/second)	Position(counts)
$v(t) = -1000(t-1)^2 + 1000$	$p(t) = \int (-1000(t-1)^2 + 1000)dt$
$v(.25) = 437.5$	p(0 to .25) = 57
$v(.5) = 750$	p(.25 to .5) = 151
$v(.75) = 937.5$	p(.5 to .75) = 214
$v(1) = 1000$	p(.75 to 1) = 245
$v(1.25) = 937.5$	p(1 to 1.25) = 245
$v(1.5) = 750$	p(1.25 to 1.5) = 214
$v(1.75) = 437.5$	p(1.5 to 1.75) = 151
$v(2) = 0$	p(1.75 to 2) =57

The DMC program is shown below and the results can be seen in Figure 6.16.

INSTRUCTION	INTERPRETATION
`#PVT`	Label
`PVX = 57,437,256`	Incremental move of 57 counts in 256 samples with a final velocity of 437 counts/sec
`PVX = 151,750,256`	Incremental move of 151 counts in 256 samples with a final velocity of 750 counts/sec
`PVX = 214,937,256`	Incremental move of 214 counts in 256 samples with a final velocity of 937 counts/sec
`PVX = 245,1000,256`	Incremental move of 245 counts in 256 samples with a final velocity of 1000 counts/sec
`PVX = 245,937,256`	Incremental move of 245 counts in 256 samples with a final velocity of 937 counts/sec
`PVX = 214,750,256`	Incremental move of 214 counts in 256 samples with a final velocity of 750 counts/sec
`PVX = 151,437,256`	Incremental move of 151 counts in 256 samples with a final velocity of 437 counts/sec
`PVX = 57,0,256`	Incremental move of 57 counts in 256 samples with a final velocity of 0 counts/sec
`PVX = 0,0,0`	Termination of PVT buffer
`BTX`	Begin PVT
`EN`	

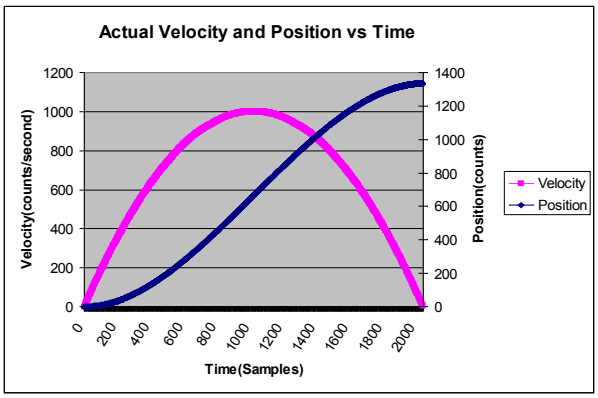


Figure 6.16: Actual Velocity and Position vs Time of Parabolic Velocity Profile

Multi-Axis Coordinated Move

Many applications require moving two or more axes in a coordinated move yet still require smooth motion at the same time. These applications are ideal candidates for PVT mode.

X Axis	Y Axis
500	500
1500	5000
2500	4000
3300	4200
7300	3300

Figure 6.17: Required X,Y points for both axes.

In this example we will have a 2 dimensional stage that needs to follow a specific profile. The application requires that the certain points be met however the path between points is not important. Smooth motion between points is critical.

The resultant DMC program is shown below. The position points are dictated by the application requirements and the velocities and times were chosen to create smooth yet quick motion. For example, in the second segment the B axis is slowed to 0 at the end of the move in anticipation of reversing direction during the next segment.

INSTRUCTION	INTERPRETATION
#PVT	Label
PVA = 500,2000,500	1st point in Figure 6.17 - A axis
PVB = 500,5000,500	1st point in Figure 6.17 - B axis
PVA = 1000,4000,1200	2nd point in Figure 6.17 - A axis
PVB = 4500,0,1200	2nd point in Figure 6.17 - B axis
PVA = 1000,4000,750	3rd point in Figure 6.17 - A axis
PVB = -1000,1000,750	3rd point in Figure 6.17 - B axis
BTAB	Begin PVT mode for A and B axes
PVA = 800,10000,250	4th point in Figure 6.17 - A axis
PVB = 200,1000,250	4th point in Figure 6.17 - B axis
PVA = 4000,0,1000	5th point in Figure 6.17 - A axis
PVB = -900,0,1000	5th point in Figure 6.17 - B axis
PVA = 0,0,0	Termination of PVT buffer for A axis
PVB = 0,0,0	Termination of PVT buffer for B axis
EN	

NOTE: The BT command is issued prior to filling the PVT buffers and additional PV commands are added during motion for demonstration purposes only. The BT command could have been issued at the end of all the PVT points in this example.

The resultant X vs. Y position graph is shown in Figure 6.18, with the specified PVT points enlarged.

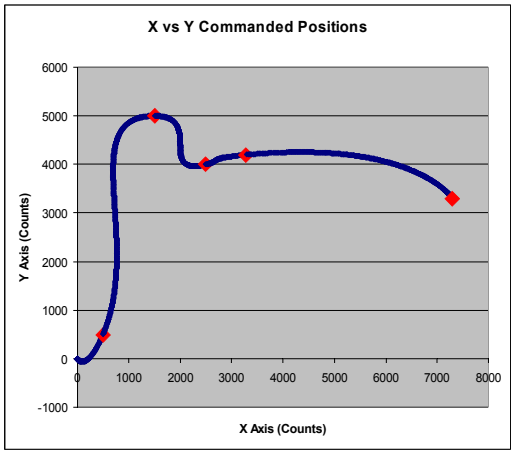


Figure 6.18: X vs Y Commanded Positions for Multi-Axis Coordinated Move

Contour Mode

The DMC-42x0 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1 to 8 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

The Contour Mode is specified with the command, CM. For example, CMXZ specifies contouring on the X and Z axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour is described by position increments which are described with the command, CD x,y,z,w over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n sample period (1 ms for TM1000), where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each sample. If the time interval changes for each segment, use CD x,y,z,w=n where n is the new DT value.

Consider, for example, the trajectory shown in Figure 6.19. The position X may be described by the points:

Point 1	X=0 at T=0ms
Point 2	X=48 at T=4ms
Point 3	X=288 at T=12ms
Point 4	X=336 at T=28ms

The same trajectory may be represented by the increments

Increment 1	DX=48	Time=4	DT=2
Increment 2	DX=240	Time=8	DT=3
Increment 3	DX=48	Time=16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

`#A`	
`CMX`	Specifies X axis for contour mode
`CD 48=2`	Specifies first position increment and time interval, 2^2 ms
`CD 240=3`	Specifies second position increment and time interval, 2^3 ms
`CD 48=4`	Specifies the third position increment and time interval, 2^4 ms
`CD 0=0`	End Contour buffer
`#Wait;JP#Wait,_CM<>511`	Wait until path is done
`EN`	

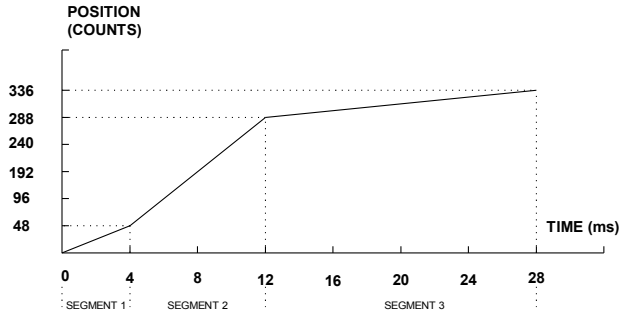


Figure 6.19: The Required Trajectory

Additional Information

_CM gives the amount of space available in the contour buffer (511 maximum). Zero parameters for DT followed by zero parameters for CD will exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Specifying a -1 for the DT or as the time interval in the CD command will pause the contour buffer.

Issuing the CM command will clear the contour buffer.

Command Summary - Contour Mode

COMMAND	DESCRIPTION
CM XYZW	Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes.
CM ABCDEFGH	Contour axes for DMC-4280
CD x,y,z,w	Specifies position increment over time interval. Range is ±32,000. CD 0,0,0.. .=0 ends the contour buffer. This is much like the LE or VE commands.
CD a,b,c,d,e,f,g,h	Position increment data for DMC-4280
DT n	Specifies time interval 2^n sample periods (1 ms for TM1000) for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD.
_CM	Amount of space left in contour buffer (511 maximum)

General Velocity Profiles

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Generating an Array - An Example

Consider the velocity and position profiles shown in Figure 6.20. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\omega = \frac{\mathrm{A}}{\mathrm{B}}\left(1 - \cos(2\pi/\mathrm{B})\right)$$

$$\mathrm{X} = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi/B)$$

Note: ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

X = 50T - (6000/2π) sin (2πT/120)

Note that the velocity, ω, in count/ms, is

ω = 50 [1 - cos 2πT/120]

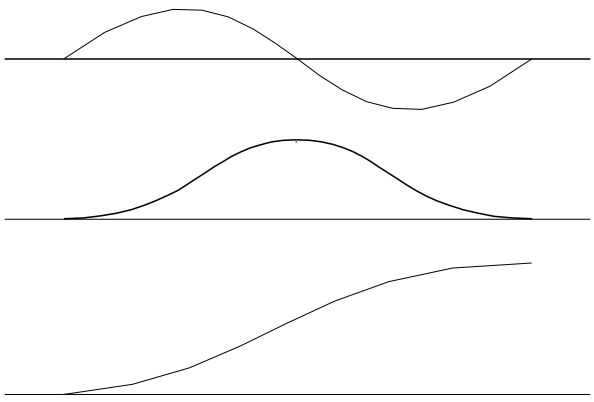


Figure 6.20: Velocity Profile with Sinusoidal Acceleration

The DMC-42x0 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

X = 50T - 955 sin 3T

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

Contour Mode Example

INSTRUCTION	INTERPRETATION
`#POINTS`	Program defines X points
`DM POS[16]`	Allocate memory
`DM DIF[15]`	
`C=0`	Set initial conditions, C is index
`T=0`	T is time in ms
`#A`	
`V1=50*T`	
`V2=3*T`	Argument in degrees
`V3=-955*@SIN[V2]+V1`	Compute position
`V4=@INT[V3]`	Integer value of V3
`POS[C]=V4`	Store in array POS
`T=T+8`	
`C=C+1`	
`JP #A,C<16`	
`#B`	Program to find position differences
`C=0`	
`#C`	
`D=C+1`	
`DIF[C]=POS[D]-POS[C]`	Compute the difference and store
`C=C+1`	
`JP #C,C<15`	
`#RUN`	Program to run motor
`CMX`	Contour Mode
`DT3`	8 millisecond intervals
`C=0`	
`#E`	
`CD DIF[C]`	Contour Distance is in DIF
`C=C+1`	
`JP #E,C<15`	
`CD 0=0`	End contour buffer
`#Wait;JP#Wait,_CM<>511`	Wait until path is done
`EN`	End the program

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the DMC-42x0 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

`DM C[n]`	Dimension array
`RA C[]`	Specify array for automatic record (up to 4 for DMC-4240)
`RD _TPX`	Specify data for capturing (such as _TPX or _TPZ)
`RC n,m`	Specify capture time interval where n is 2^n sample periods (1 ms for TM1000), m is number of records to be captured
`RC? or _RC`	Returns a 1 if recording

Record and Playback Example:

```
#RECORD                  Begin Program
DM XPOS[501]             Dimension array with 501 elements
RA XPOS[]                Specify automatic record
RD _TPX                  Specify X position to be captured
MOX                      Turn X motor off
RC2                      Begin recording; 4 msec interval (at TM1000)
#A;JP#A,_RC=1            Continue until done recording
#COMPUTE                 Compute DX
DM DX[500]               Dimension Array for DX
C=0                      Initialize counter
#L                       Label
D=C+1
DELTA=XPOS[D]-XPOS[C]    Compute the difference
DX[C]=DELTA              Store difference in array
C=C+1                    Increment index
JP #L,C<500              Repeat until done
#PLAYBCK                 Begin Playback
CMX                      Specify contour mode
DT2                      Specify time increment
I=0                      Initialize array counter
#B                       Loop counter
CD DX[I]; I=I+1          Specify contour data I=I+1 Increment array counter
JP #B,I<500              Loop until done
CD 0=0                   End countour buffer
#Wait;JP#Wait,_CM<>511   Wait until path is done
EN                       End program
```

For additional information about automatic array capture, see Chapter 7, Arrays.

Virtual Axis

The DMC-42x0 controller has two additional virtual axes designated as the M and N axes. These axes have no encoder and no DAC. However, they can be commanded by the commands:

AC, DC, JG, SP, PR, PA, BG, IT, GA, VM, VP, CR, ST, DP, RP

The main use of the virtual axes is to serve as a virtual master in ECAM modes, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

ECAM Master Example

Suppose that the motion of the XY axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN= 4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands.

For example,

```
PAN = 2000
BGN
```

will cause the XY axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The x axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20 Hz.

This can be performed by commanding the X and N axes to perform circular motion. Note that the value of VS must be

$$VS=2\pi * R * F$$

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration.

INSTRUCTION	INTERPRETATION
VMXN	Select Axes
VA 68000000	Maximum Acceleration
VD 68000000	Maximum Deceleration
VS 125664	VS for 20 Hz
CR 1000, -90, 3600	Ten Cycles
VE	
BGS	

Stepper Motor Operation

When configured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors.

Specifying Stepper Motor Operation

Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2 specifies a stepper motor with active low step output pulses
-2 specifies a stepper motor with active high step output pulses
2.5 specifies a stepper motor with active low step output pulses and reversed direction
-2.5 specifies a stepper motor with active high step output pulse and reversed direction

Stepper Motor Smoothing

The command, KS, provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since this filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, Monitoring Generated Pulses vs. Commanded Pulses.

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease ‘jerk’ due to acceleration.

Monitoring Generated Pulses vs. Commanded Pulses

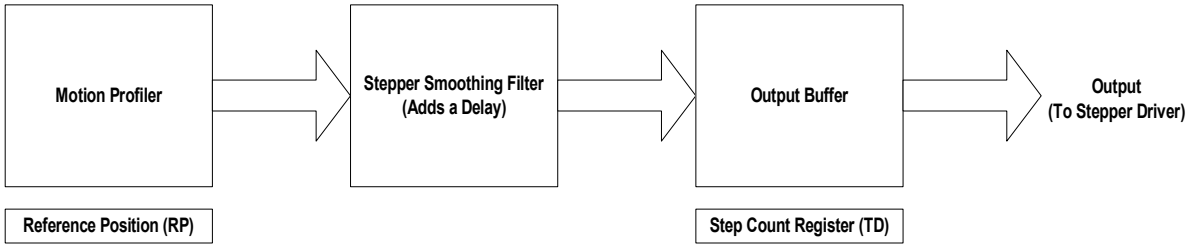
For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion profiler is complete and is prepared to execute a new motion command. However when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter which is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses which are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.



Motion Complete Trippoint

When used in stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input. Note: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

Note: Closed loop operation with a stepper motor is not possible.

Command Summary - Stepper Motor Operation

COMMAND	DESCRIPTION
DE	Define Encoder Position (When using an encoder)
DP	Define Reference Position and Step Count Register
IT	Motion Profile Smoothing - Independent Time Constant
KS	Stepper Motor Smoothing
MT	Motor Type (2,-2,2.5 or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

OPERAND	DESCRIPTION
_DEx	Contains the value of the step count register for the 'x' axis
_DPx	Contains the value of the main encoder for the 'x' axis
_ITx	Contains the value of the Independent Time constant for the 'x' axis
_KSx	Contains the value of the Stepper Motor Smoothing Constant for the 'x' axis
_MTx	Contains the motor type value for the 'x' axis
_RPx	Contains the commanded position generated by the profiler for the 'x' axis
_TDx	Contains the value of the step count register for the 'x' axis
_TPx	Contains the value of the main encoder for the 'x' axis

Stepper Position Maintenance Mode (SPM)

The Galil controller can be set into the Stepper Position Maintenance (SPM) mode to handle the event of stepper motor position error. The mode looks at position feedback from the main encoder and compares it to the commanded step pulses. The position information is used to determine if there is any significant difference between the commanded and the actual motor positions. If such error is detected, it is updated into a command value for operator use. In addition, the SPM mode can be used as a method to correct for friction at the end of a microstepping move. This capability provides closed-loop control at the application program level. SPM mode can be used with Galil and non-Galil step drives.

SPM mode is configured, executed, and managed with seven commands. This mode also utilizes the #POSERR automatic subroutine allowing for automatic user-defined handling of an error event.

Internal Controller Commands (user can query):

QS Error Magnitude (pulses)

User Configurable Commands (user can query & change):

OE Profiler Off-On Error
YA Step Drive Resolution (pulses / full motor step)
YB Step Motor Resolution (full motor steps / revolution)
YC Encoder Resolution (counts / revolution)
YR Error Correction (pulses)
YS Stepper Position Maintenance enable, status

A pulse is defined by the resolution of the step drive being used. Therefore, one pulse could be a full step, a half step or a microstep.

When a Galil controller is configured for step motor operation, the step pulse output by the controller is internally fed back to the auxiliary encoder register. For SPM the feedback encoder on the stepper will connect to the main encoder port. Enabling the SPM mode on a controller with YS=1 executes an internal monitoring of the auxiliary and main encoder registers for that axis or axes. Position error is then tracked in step pulses between these two registers (QS command).

$$QS = TD - \frac{TP \times YA \times YB}{YC}$$

Where TD is the auxiliary encoder register(step pulses) and TP is the main encoder register(feedback encoder). Additionally, YA defines the step drive resolution where YA = 1 for full stepping or YA = 2 for half stepping. The full range of YA is up to YA = 9999 for microstepping drives.

Error Limit

The value of QS is internally monitored to determine if it exceeds a preset limit of three full motor steps. Once the value of QS exceeds this limit, the controller then performs the following actions:

1. The motion is maintained or is stopped, depending on the setting of the OE command. If OE=0 the axis stays in motion, if OE=1 the axis is stopped.
2. YS is set to 2, which causes the automatic subroutine labeled #POSERR to be executed.

Correction

A correction move can be commanded by assigning the value of QS to the YR correction move command. The correction move is issued only after the axis has been stopped. After an error correction move has completed and QS is less than three full motor steps, the YS error status bit is automatically reset back to 1; indicating a cleared error.

Example: SPM Mode Setup

The following code demonstrates what is necessary to set up SPM mode for a full step drive, a half step drive, and a 1/64th microstepping drive for an axis with a 1.8° step motor and 4000 count/rev encoder. Note the necessary difference is with the YA command.

Full-Stepping Drive, X axis:

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA1;          Step resolution of the full-step drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
YS1;          Enable SPM mode
```

Half-Stepping Drive, X axis:

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA2;          Step resolution of the half-step drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
YS1;          Enable SPM mode
```

1/64th Step Microstepping Drive, X axis:

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA64;         Step resolution of the microstepping drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
YS1;          Enable SPM mode
```

Example: Error Correction

The following code demonstrates what is necessary to set up SPM mode for the X axis, detect error, stop the motor, correct the error, and return to the main code. The drive is a full step drive, with a 1.8° step motor and 4000 count/rev encoder.

```
#setup
OE 1;'                  Set the profiler to stop axis upon error
KS 16;'                 Set step smoothing
MT -2,-2,-2,-2;'        Motor type set to stepper
YA 2;'                  Step resolution of the drive
YB 200;'                Motor resolution (full steps per revolution)
YC 4000;'               Encoder resolution (counts per revolution)
SH A;'                  Enable axis
WT 100;'                Allow slight settle time


#motion;'               Perform motion
SP 512;'                Set the speed
PR 1000;'               Prepare mode of motion
BG A;'                  Begin motion
EN;'                    End of program subroutine

REM  When error occurs, the axis will stop due to OE1.  In
REM #POSERR, query the status YS and the error QS, correct,
REM and return to the main code.

#POSERR;'               Automatic subroutine is called when _YS=2
WT 100;'                Wait helps user see the correction
spsave=_SPA;'           Save current speed setting
JP #return,_YSA<>2;'    Return to thread zero if invalid error
SP64;                   Set slow speed setting for correction
MG "ERROR= ",_QSA
YRA=_QSA;'              Else, error is valid, use QS for correction
MC A;                   Wait for motion to complete
MG "CORRECTED, ERROR NOW= ",_QSX
WT 100;'                Wait helps user see the correction

#return
SPA=spsave;'            Return the speed to previous setting
RE 0;'                  Return from #POSERR
```

Example: Friction Correction

The following example illustrates how the SPM mode can be useful in correcting for X axis friction after each move when conducting a reciprocating motion. The drive is a 1/64th microstepping drive with a 1.8° step motor and 4000 count/rev encoder.

```
#SETUP;                 Set the profiler to continue upon error
KS16;                   Set step smoothing
MT-2,-2,-2,-2;          Motor type set to stepper
YA64;                   Step resolution of the microstepping drive
YB200;                  Motor resolution (full steps per revolution)
YC4000;                 Encoder resolution (counts per revolution)
SHX;                    Enable axis
WT50;                   Allow slight settle time
YS1;                    Enable SPM mode

#MOTION;                Perform motion
SP16384;                Set the speed
PR10000;                Prepare mode of motion
BGX;                    Begin motion
MCX
JS#CORRECT;             Move to correction
#MOTION2
SP16384;                Set the speed
PR-10000;               Prepare mode of motion
BGX;                    Begin motion
MCX
JS#CORRECT;             Move to correction
JP#MOTION
#CORRECT;               Correction code
spx=_SPX
#LOOP;                  Save speed value
SP2048;                 Set a new slow correction speed
WT100;                  Stabilize
JP#END,@ABS[_QSX]<10;   End correction if error is within defined tolerance
YRX=_QSX;               Correction move
MCX
WT100;                  Stabilize
JP#LOOP;                Keep correcting until error is within tolerance
#END;                   End #CORRECT subroutine, returning to code
SPX=spx
EN
```

Dual Loop (Auxiliary Encoder)

The DMC-42x0 provides an interface for a second encoder for each axis except for axes configured for stepper motor operation and axis used in circular compare. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE x,y,z,w (or a,b,c,d,e,f,g,h for controllers with more than 4 axes) where the parameters x,y,z,w each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

Using the CE Command

m=	Main Encoder	n=	Second Encoder
0	Normal quadrature	0	Normal quadrature
1	Pulse & direction	4	Pulse & direction
2	Reverse quadrature	8	Reversed quadrature
3	Reverse pulse & direction	12	Reversed pulse & direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the X axis is:

```
CE 6
```

Additional Commands for the Auxiliary Encoder

The command, DE x,y,z,w, can be used to define the position of the auxiliary encoders. For example,

```
DE 0,500,-30,300
```

sets their initial values. The positions of the auxiliary encoders may be interrogated with the command, DE?. For example:

```
DE ?,,?
```

returns the value of the X and Z auxiliary encoders.

The auxiliary encoder position may be assigned to variables with the instructions

```
V1= _DEX
```

The command, TD XYZW, returns the current position of the auxiliary encoder.

The command, DV 1,1,1,1, configures the auxiliary encoder to be used for backlash compensation.

Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoders:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

Continuous Dual Loop - Example

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

```
DV    1,1,1,1
```

activates the dual loop for the four axes and

```
DV      0,0,0,0
```

disables the dual loop.

Note: that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled Dual Loop - Example

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the X-axis and connect the linear encoder to the auxiliary encoder of X. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

INSTRUCTION	INTERPRETATION
`#DUALOOP`	Label
`CE 0`	Configure encoder
`DE0`	Set initial value
`PR 40000`	Main move
`BGX`	Start motion
`#correct`	Correction loop
`AMX`	Wait for motion completion
`V1=10000-_DEX`	Find linear encoder error
`V2=-_TEX/4+V1`	Compensate for motor error
`JP#END,@ABS[V2]<2`	Exit if error is small
`PR V2*4`	Correction move
`BGX`	Start correction
`JP#correct`	Repeat
`#END`	
`EN`	

Motion Smoothing

The DMC-42x0 controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

Using the IT Command:

When operating with servo motors, motion smoothing can be accomplished with the IT command. This command filters the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

`IT x,y,z,w`	Independent time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and to smooth vector moves of the type VM and LM.

The smoothing parameters, x,y,z,w and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Figure 6.21 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example - Smoothing

`PR 20000`	Position
`AC 100000`	Acceleration
`DC 100000`	Deceleration
`SP 5000`	Speed
`IT .5`	Filter for smoothing
`BG X`	Begin

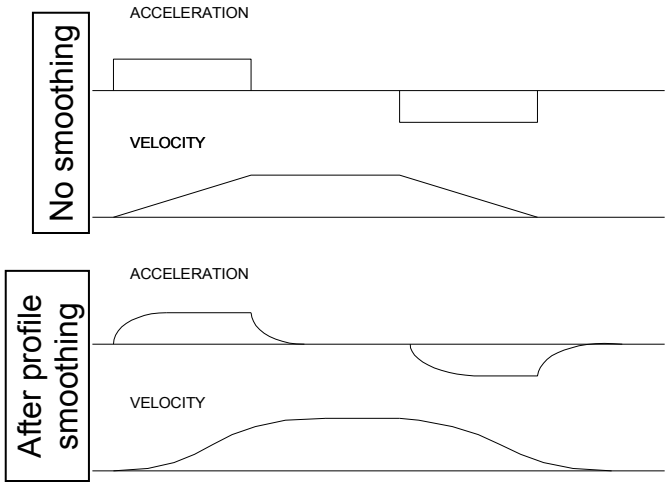


Figure 6.21: Trapezoidal velocity and smooth velocity profiles

Using the KS Command (Step Motor Smoothing):

When operating with step motors, motion smoothing can be accomplished with the command, KS. The KS command smoothes the frequency of step motor pulses. Similar to the command IT, this produces a smooth velocity profile.

The step motor smoothing is specified by the following command:

`KS x,y,z,w`	where x,y,z,w is an integer from 0.25 to 64 and represents the amount of smoothing

The smoothing parameters, x,y,z,w and n are numbers between 0.25 and 64 and determine the degree of filtering. The minimum value of 0.25 implies the least filtering, resulting in trapezoidal velocity profiles. Larger values of the smoothing parameters imply heavier filtering and smoother moves.

Note that KS is valid only for step motors.

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The HM command and BG command causes the following sequence of events to occur.

Stage 1:

Upon begin, the motor accelerates to the slew speed specified by the JG or SP commands. The direction of its motion is determined by the state of the homing input. If _HMX reads 1 initially, the motor will go in the reverse direction first (direction of decreasing encoder counts). If _HMX reads 0 initially, the motor will go in the forward direction first. CN is the command used to define the polarity of the home input. With CN,-1 (the default value) a normally open switch will make _HMX read 1 initially, and a normally closed switch will make _HMX read zero. Furthermore, with CN,1 a normally open switch will make _HMX read 0 initially, and a normally closed switch will make _HMX read 1. Therefore, the CN command will need to be configured properly to ensure the correct direction of motion in the home sequence.

Upon detecting the home switch changing state, the motor begins decelerating to a stop.

Note: The direction of motion for the FE command also follows these rules for the state of the home input.

Stage 2:

The motor then traverses at HV counts/sec in the opposite direction of Stage 1 until the home switch toggles again. If Stage 3 is in the opposite direction of Stage 2, the motor will stop immediately at this point and change direction. If Stage 2 is in the same direction as Stage 3, the motor will never stop, but will smoothly continue into Stage 3.

Stage 3:

The motor traverses forward at HV counts/sec until the encoder index pulse is detected. The motor then decelerates to a stop and goes back to the index.

The DMC-42x0 defines the home position as the position at which the index was detected and sets the encoder reading at this point to zero.

The 4 different motion possibilities for the home sequence are shown in the following table.

			Direction of Motion		
Switch Type	**CN Setting**	**Initial _HMX state**	**Stage 1**	**Stage 2**	**Stage 3**
Normally Open	CN,-1	1	Reverse	Forward	Forward
Normally Open	CN,1	0	Forward	Reverse	Forward
Normally Closed	CN,-1	0	Forward	Reverse	Forward
Normally Closed	CN,1	1	Reverse	Forward	Forward

Example: Homing

Instruction	Interpretation
#HOME	Label
CN,-1	Configure the polarity of the home input
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
SP 5000	Speed for Home Search
HM	Home
BG	Begin Motion
AM	After Complete
MG "AT HOME"	Send Message
EN	End

Figure 6.22 shows the velocity profile from the homing sequence of the example program above. For this profile, the switch is normally closed and CN,-1.

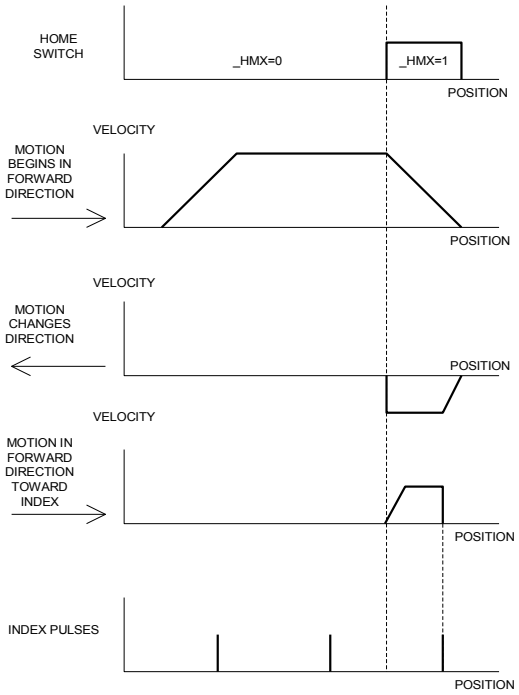


Figure 6.22: Homing Sequence for Normally Closed Switch and CN,-1

Example: Find Edge

#EDGE	Label
AC 2000000	Acceleration rate
DC 2000000	Deceleration rate
SP 8000	Speed
FE	Find edge command
BG	Begin motion
AM	After complete
MG "FOUND HOME"	Send message
DP 0	Define position as 0
EN	End

Command Summary - Homing Operation

Command	Description
FE XYZW	Find Edge Routine. This routine monitors the Home Input
FI XYZW	Find Index Routine - This routine monitors the Index Input
HM XYZW	Home Routine - This routine combines FE and FI as Described Above
SC XYZW	Stop Code
TS XYZW	Tell Status of Switches and Inputs

Operand Summary - Homing Operation

Operand	Description
_HMx	Contains the value of the state of the Home Input
_SCx	Contains stop code
_TSx	Contains status of switches and inputs

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. Position capture can be programmed to latch on either a corresponding input (see Table 6.13) or on the index pulse for that axis. The position can be captured for either the main or auxiliary encoder within 25 microseconds of an high-to-low transition.

Input 1	A-axis latch	Input 9	E-axis latch
Input 2	B-axis latch	Input 10	F-axis latch
Input 3	C-axis latch	Input 11	G-axis latch
Input 4	D-axis latch	Input 12	H-axis latch

Table 6.13: Inputs and corresponding axis latch

NOTE	Latching is not valid with sampled feedback types such as: SSI, BiSS, Sin/Cos, and Analog

To insure a position capture within 25 microseconds, the input signal must be a transition from high to low. Low to high transitions may have greater delay.

The software commands, AL and RL, are used to arm the latch and report the latched position respectively. The latch must be re-armed after each latching event. See the Command Reference for more details on these commands.

Chapter 7 Application Programming

Overview

The DMC-42x0 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-42x0 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the DMC-42x0 provides commands that allow the DMC-42x0 to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the DMC-42x0 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs. The program memory size is 80 characters x 4000 lines.

Program Format

A DMC-42x0 program consists of DMC instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each DMC-42x0 instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 80 characters. A carriage return enters the final command on a program line.

Using Labels in Programs

All DMC-42x0 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted in label.

The maximum number of labels which may be defined is 510.

Valid labels

```
#BEGIN
#SQUARE
```

```
#X1
#BEGIN1
```

Invalid labels

```
#1Square
#123
```

A Simple Example Program:

```
#START                  Beginning of the Program
PR 10000,20000          Specify relative distances on X and Y axes
BG XY                   Begin Motion
AM                      Wait for motion complete
WT 2000                 Wait 2 sec
JP #START               Jump to label START
EN                      End of Program
```

The above program moves X and Y 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

Special Labels

The DMC-42x0 have some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. See section on ___

#AMPERR	Label for Amplifier error routine
#AUTO	Label that will automatically run upon the controller exiting a reset (power-on)
#AUTOERR	Label that will automatically run if there is an EEPROM error out of reset
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for Communications Interrupt (See CC Command)
#ININT	Label for Input Interrupt subroutine (See II Command)
#LIMSWI	Label for Limit Switch subroutine
#MCTIME	Label for timeout on Motion Complete trippoint
#POSERR	Label for excess Position Error subroutine
#TCPERR	Label for errors over a TCP connection (error code 123)

Commenting Programs

Using the command, NO or Apostrophe (‘)

The DMC-42x0 provides a command, NO, for commenting programs or single apostrophe. This command allows the user to include up to 78 characters on a single line after the NO command and can be used to include comments from the programmer as in the following example:

```
#PATH
' 2-D CIRCULAR PATH
VMXY
'  VECTOR MOTION ON X AND Y
VS 10000
'  VECTOR SPEED IS 10000
VP -4000,0
'  BOTTOM LINE
CR 1500,270,-180
'  HALF CIRCLE MOTION
VP 0,3000
'  TOP LINE
CR 1500,90,-180
'  HALF CIRCLE MOTION
VE
'  END VECTOR SEQUENCE
BGS
'  BEGIN SEQUENCE MOTION
EN
'  END OF PROGRAM
```

Note: The NO command is an actual controller command. Therefore, inclusion of the NO commands will require process time by the controller.

Difference between NO and ' using the GalilTools software

The GalilTools software will treat an apostrophe (') commend different from an NO when the compression algorithm is activated upon a program download (line > 80 characters or program memory > 4000 lines). In this case the software will remove all (') comments as part of the compression and it will download all NO comments to the controller.

Executing Programs - Multitasking

The DMC-42x0 can run up to 8 independent programs simultaneously. These programs are called threads and are numbered 0 through 7, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. Only the main thread, thread 0, may use the input command, IN.
2. When input interrupts are implemented for limit switches, position errors or command errors, the subroutines are executed as thread 0.

To begin execution of the various programs, use the following instruction:

```
XQ #A, n
```

Where n indicates the thread number. To halt the execution of any thread, use the instruction

```
HX n
```

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

`#TASK1`	Task1 label
`AT0`	Initialize reference time
`CB1`	Clear Output 1
`#LOOP1`	Loop1 label
`AT 10`	Wait 10 msec from reference time
`SB1`	Set Output 1
`AT -40`	Wait 40 msec from reference time, then initialize reference
`CB1`	Clear Output 1
`JP #LOOP1`	Repeat Loop1
`#TASK2`	Task2 label
`XQ #TASK1,1`	Execute Task1
`#LOOP2`	Loop2 label
`PR 1000`	Define relative distance
`BGX`	Begin motion
`AMX`	After motion done
`WT 10`	Wait 10 msec
`JP #LOOP2,@IN[2]=1`	Repeat motion unless Input 2 is low
`HX`	Halt all tasks

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (i.e. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

The DMC-42x0 provides commands and operands which are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and the contents of the controllers program, array, and variable space. Operands also contain important status information which can help to debug a program.

Trace Commands

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off. Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

NOTE: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Data which is output from the controller is stored in the output UART. The UART buffer can store up to 512 characters of information. In normal operation, the controller places output into the FIFO buffer. When the trace mode is enabled, the controller will send information to the UART buffer at a very high rate. In general, the UART will become full because the hardware handshake line will halt serial data until the correct data is read. When the UART becomes full, program execution will be delayed until it is cleared. If the user wants to avoid this delay, the command CW,1 can be given. This command causes the controller to throw away the data which can not be placed into the FIFO. In this case, the controller does not delay program execution.

Error Code Command

When there is a program error, the DMC-42x0 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

RAM Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-42x0 has several useful commands. The command, DM ?, will return the number of array elements currently available. The command, DA ?, will return the number of arrays which can be currently defined. For example, a standard DMC-14010 will have a maximum of 24000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 15900 and the command DA ? will return 29.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

Operands

In general, all operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.
_DL contains the number of available labels.
_UL contains the number of available variables.
_DA contains the number of available arrays.
_DM contains the number of available array elements.
_AB contains the state of the Abort Input
_LFx contains the state of the forward limit switch for the 'x' axis
_LRx contains the state of the reverse limit switch for the 'x' axis

Debugging Example:

The following program has an error. It attempts to specify a relative movement while the X-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

```
Download Code
#A                      Program Label
PR1000                  Position Relative 1000
BGX                     Begin
PR5000                  Position Relative 5000
EN                      End
From Terminal
:XQ #A                  Execute #A
?003 PR5000             Error on Line 3
:TC1                    Tell Error Code
?7 Command not valid    Command not valid while running
while running.
                        Change the BGX line to BGX;AMX and re-download the program.
:XQ #A                  Execute #A
```

Program Flow Commands

The DMC-42x0 provides instructions to control program flow. The controller program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

Event Triggers & Trippoints

To function independently from the host computer, the DMC-42x0 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The DMC-42x0 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the controller can make decisions based on its own status or external events without intervention from a host computer.

DMC-42x0 Event Triggers

Command	Function
AM X Y Z W or S (A B C D E F G H)	Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes. This command is useful for separating motion sequences in a program.
AD X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time.
AP X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
MF X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x,y,z,w sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stop code will be set to 99. An application program will jump to label #MCTIME.
AI ± n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 8 for DMC-4210, 4020, 4030, 4040. n=1 through 16 for DMC-4250, 4060, 4070, 4080 Also n= 17-48
AS X Y Z W S (A B C D E F G H)	Halts program execution until specified axis has reached its slew speed.
AT ±n,m	For m=omitted or 0, halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time. For m=1. Same functionality except that n is number of samples rather than msec
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n,m	For m=omitted or 0, halts program execution until specified time in msec has elapsed. For m=1. Same functionality except that n is number of samples rather than msec.

Event Trigger Examples:

Event Trigger - Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

```
#TWOMOVE;'          Label
PR 2000;'           Position Command
BGX;'               Begin Motion
AMX;'               Wait for Motion Complete
PR 4000;'           Next Position Move
BGX;'               Begin 2nd move
EN;'                End program
```

Event Trigger - Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

```
#SETBIT;'              Label
SP 10000;'             Speed is 10000
PA 20000;'             Specify Absolute position
BGX;'                  Begin motion
AD 1000;'              Wait until 1000 counts
SB1;'                  Set output bit 1
EN;'                   End program
```

Event Trigger - Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

```
#TRIP;'                Label
JG 50000;'             Specify Jog Speed
BGX;n=0;'              Begin Motion
#REPEAT;'              # Repeat Loop
AR 10000;'             Wait 10000 counts
TPX;'                  Tell Position
SB1;'                  Set output 1
WT50;'                 Wait 50 msec
CB1;'                  Clear output 1
n=n+1;'                Increment counter
JP #REPEAT,n<5;'       Repeat 5 times
STX;'                  Stop
EN;'                   End
```

Event Trigger - Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP#GO,@IN[1] = 1.

```
#INPUT;'               Program Label
AI-1;'                 Wait for input 1 low
PR 10000;'             Position command
BGX;'                  Begin motion
EN;'                   End program
```

Event Trigger - Set output when At speed

```
#ATSPEED;'             Program Label
JG 50000;'             Specify jog speed
AC 10000;'             Acceleration rate
BGX;'                  Begin motion
ASX;'                  Wait for at slew speed 50000
SB1;'                  Set output 1
EN;'                   End program
```

Event Trigger - Change Speed along Vector Path

The following program changes the feed rate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

```
#VECTOR;'              Label
VMXY;VS 5000;'         Coordinated path
VP 10000,20000;'       Vector position
VP 20000,30000;'       Vector position
VE;'                   End vector
BGS;'                  Begin sequence
AV 5000;'              After vector distance
VS 1000;'              Reduce speed
EN;'                   End
```

Event Trigger - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

```
#MOVES;'            Label
PR 12000;'          Distance
SP 20000;'          Speed
AC 100000;'         Acceleration
BGX;'               Start Motion
AD 10000;'          Wait a distance of 10,000 counts
SP 5000;'           New Speed
AMX;'               Wait until motion is completed
WT 200;'            Wait 200 ms
PR -10000;'         New Position
SP 30000;'          New Speed
AC 150000;'         New Acceleration
BGX;'               Start Motion
EN;'                End
```

Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

```
#OUTPUT;'           Program label
AT0;'               Initialize time reference
SB1;'               Set Output 1
#LOOP;'             Loop
AT 10;'             After 10 msec from reference,
CB1;'               Clear Output 1
AT -40;'            Wait 40 msec from reference and reset reference
SB1;'               Set Output 1
JP #LOOP;'          Loop
EN;'                End Program
```

Using AT/WT with non-default TM rates

By default both WT and AT are defined to hold up program execution for 'n' number of milliseconds (WT n or AT n). The second field of both AT and WT can be used to have the program execution be held-up for 'n' number of samples rather than milliseconds. For example WT 400 or WT 400,0 will hold up program execution for 400 msec regardless of what is set for TM. By contrast WT 400,1 will hold up program execution for 400 samples. For the default TM of 1000 the servo update rate is 976us per sample, so the difference between WT n,0 and WT n,1 is minimal. The difference comes when the servo update rate is changed. With a low servo update rate, it is often useful to be able to time loops based upon samples rather than msec, and this is where the “unscaled” WT and AT are useful. For example:

```
#MAIN;'             Label
TM 250;'            250us update rate
#MOVE;'             Label
PRX=1000;'          Position Relative Move
BGX;'               Begin Motion
MCX;'               Wait for motion to complete
WT 2,1;'            Wait 2 samples (500us)
SB1;'               Set bit 1
EN;'                End Program
```

In the above example, without using an unscaled WT, the output would either need to be set directly after the motion was complete, or 2 ms after the motion was complete. By using WT n,1 and a lower TM, greater delay resolution was achieved.

Conditional Jumps

The DMC-42x0 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. The conditional jump determines if a condition is satisfied and then branches to a new location or subroutine. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Conditional jumps are useful for testing events in real-time. They allow the controller to make decisions without a host computer. For example, the DMC-42x0 can decide between two motion profiles based on the state of an input line.

Command Format - JP and JS

FORMAT	DESCRIPTION
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

Logical operators:

OPERATOR	DESCRIPTION
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal

Conditional Statements

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid DMC-42x0 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

Examples:

Number	`v1=6`
Numeric Expression	`v1=v7*6`
	`@ABS[v1]>10`
Array Element	`v1<count[2]`
Variable	`v1<v2`
Internal Variable	`_TPX=0`
	`_TVX>500`
I/O	`v1>@AN[2]`
	`@IN[1]=0`

Multiple Conditional Statements

The DMC-42x0 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

Note: Each condition must be placed in parentheses for proper evaluation by the controller. In addition, the DMC-42x0 executes operations from left to right. See Mathematical and Functional Expressions for more information.

For example, using variables named v1, v2, v3 and v4:

```
JP #TEST,((v1<v2)&(v3<v4))
```

In this example, this statement will cause the program to jump to the label #TEST if v1 is less than v2 and v3 is less than v4. To illustrate this further, consider this same example with an additional condition:

```
JP #TEST, ((v1<v2) & (v3<v4)) | (v5<v6)
```

This statement will cause the program to jump to the label #TEST under two conditions; 1. If v1 is less than v2 and v3 is less than v4. OR 2. If v5 is less than v6.

Using the JP Command:

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Conditional	Meaning
`JP #Loop,count<10`	Jump to #Loop if the variable, count, is less than 10
`JS #MOVE2,@IN[1]=1`	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
`JP #BLUE,@ABS[v2]>2`	Jump to #BLUE if the absolute value of variable, v2, is greater than 2
`JP #C,v1*v7<=v8*v2`	Jump to #C if the value of v1 times v7 is less than or equal to the value of v8*v2
`JP#A`	Jump to #A

Example Using JP command:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

`#BEGIN`	Begin Program
`count=10`	Initialize loop counter
`#LOOP`	Begin loop
`PA 1000`	Position absolute 1000
`BGX`	Begin move
`AMX`	Wait for motion complete
`WT 100`	Wait 100 msec
`PA 0`	Position absolute 0
`BGX`	Begin move
`AMX`	Wait for motion complete
`WT 100`	Wait 100 msec
`count=count-1`	Decrement loop counter
`JP #LOOP,count>0`	Test for 10 times thru loop
`EN`	End Program

Using If, Else, and Endif Commands

The DMC-42x0 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

Note: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command

and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The DMC-42x0 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-42x0 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	Description
IF conditional statement(s)	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Example using IF, ELSE and ENDIF:

```
#TEST                                   Begin Main Program "TEST"
II,,3                                   Enable input interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"       Output message
#LOOP                                   Label to be used for endless loop
JP #LOOP                                Endless loop
EN                                      End of main program
#ININT                                  Input Interrupt Subroutine
IF (@IN[1]=0)                           IF conditional statement based on input 1
IF (@IN[2]=0)                           2nd IF conditional statement executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"     Message to be executed if 2nd IF conditional is true
ELSE                                    ELSE command for 2nd IF conditional statement
MG "ONLY INPUT 1 IS ACTIVE              Message to be executed if 2nd IF conditional is false
ENDIF                                   End of 2nd conditional statement
ELSE                                    ELSE command for 1st IF conditional statement
MG"ONLY INPUT 2 IS ACTIVE"              Message to be executed if 1st IF conditional statement is false
ENDIF                                   End of 1st conditional statement
#WAIT                                   Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)         Loop until both input 1 and input 2 are not active
RI0                                     End Input Interrupt Routine without restoring trippoints
```

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

Example:

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

```
#M                      Begin Main Program
CB1                     Clear Output Bit 1 (pick up pen)
VP 1000,1000;LE;BGS     Define vector position; move pen
AMS                     Wait for after motion trippoint
SB1                     Set Output Bit 1 (put down pen)
JS #Square;CB1          Jump to square subroutine
EN                      End Main Program
#Square                 Square subroutine
v1=500;JS #L            Define length of side
v1=-v1;JS #L            Switch direction
EN                      End subroutine
#L;PR v1,v1;BGX         Define X,Y; Begin X
AMX;BGY;AMY             After motion on X, Begin Y
EN                      End subroutine
```

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

Auto-Start Routine

The DMC-42x0 has a special label for automatic program execution. A program which has been saved into the controller's non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or DMC-42x0 program sequences. The controller can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program. The pre-defined labels are:

SUBROUTINE	DESCRIPTION
#LIMSWI	Limit switch on any axis goes low
#ININT	Input specified by II goes low
#POSERR	Position error exceeds limit specified by ER
#MCTIME	Motion Complete timeout occurred. Timeout period set by TW command
#CMDERR	Bad command given
#AUTO	Automatically executes on power up
#AUTOERR	Automatically executes when a checksum is encountered during #AUTO start-up. Check error condition with _RS. bit 0 for variable checksum error bit 1 for parameter checksum error bit 2 for program checksum error bit 3 for master reset error (there should be no program)
#AMPERR	Error from internal Galil amplifier

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

NOTE: An application program must be running for #CMDERR to function.

Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the DMC-42x0 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

```
#LOOP                          Dummy Program
JP #LOOP;EN                    Jump to Loop
#LIMSWI                        Limit Switch Label
MG "LIMIT OCCURRED"            Print Message
RE                             Return to main program
                               Download Program
:XQ #LOOP                      Execute Dummy Program
:JG 5000                       Jog
:BGX                           Begin Motion
```

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

Notes regarding the #LIMSWI Routine:

1) The RE command is used to return from the #LIMSWI subroutine.
2) The #LIMSWI subroutine will be re-executed if the limit switch remains active.

The #LIMSWI routine is only executed when the motor is being commanded to move.

Example - Position Error

```
#LOOP                          Dummy Program
JP #LOOP;EN                    Loop
#POSERR                        Position Error Routine
v1=_TEX                        Read Position Error
MG "EXCESS POSITION ERROR"     Print Message
MG "ERROR=",v1=                Print Error
RE                             Return from Error
                               Download program
:XQ #LOOP                      Execute Dummy Program
:JG 100000                     Jog at High Speed
:BGX                           Begin Motion
```

Example - Input Interrupt

```
#A                             Label
II1                            Input Interrupt on 1
JG 30000,,,60000               Jog
BGXW                           Begin Motion
#LOOP;JP#LOOP;EN               Loop
#ININT                         Input Interrupt
STXW;AM                        Stop Motion
#TEST;JP #TEST, @IN[1]=0       Test for Input 1 still low
JG 30000,,,6000                Restore Velocities
BGXW                           Begin motion
RI0                            Return from interrupt routine to Main Program and do not re-enable trippoints
```

Example - Motion Complete Timeout

```
#BEGIN                  Begin main program
TW 1000                 Set the time out to 1000 ms
PA 10000                Position Absolute command
BGX                     Begin motion
MCX                     Motion Complete trippoint
EN                      End main program
#MCTIME                 Motion Complete Subroutine
MG "X fell short"       Send out a message
EN                      End subroutine
```

This simple program will issue the message "X fell short" if the X axis does not reach the commanded position within 1 second of the end of the profiled move.

Example - Command Error

```
#BEGIN                  Begin main program
speed = 2000            Set variable for speed
JG speed;BGX;           Begin motion
#LOOP
JG speed;WT100          Update Jog speed based upon speed variable
JP #LOOP
EN                      End main program
#CMDERR                 Command error utility
JP#DONE,_ED<>2          Check if error on line 2
JP#DONE,_TC<>6          Check if out of range
MG "SPEED TOO HIGH"     Send message
MG "TRY AGAIN"          Send message
ZS1                     Adjust stack
JP #BEGIN               Return to main program
#DONE                   End program if other error
ZS0                     Zero stack
EN                      End program
```

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

OPERAND	FUNCTION
_ED1	Returns the number of the thread that generated an error
_ED2	Retry failed command (operand contains the location of the failed command)
_ED3	Skip failed command (operand contains the location of the command after the failed command)

The operands are used with the XQ command in the following format:

XQ _ED2 (or _ED3),_ED1,1

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not be removed when the above format executes.

The following example shows an error correction routine which uses the operands.

Example - Command Error w/Multitasking

```
#A                          Begin thread 0 (continuous loop)
JP#A
EN                          End of thread 0

#B                          Begin thread 1
N=-1                        Create new variable
KP N                        Set KP to value of N, an invalid value
TY                          Issue invalid command
EN                          End of thread 1

#CMDERR                     Begin command error subroutine
IF _TC=6                    If error is out of range (KP -1)
N=1                         Set N to a valid number
XQ _ED2,_ED1,1              Retry KP N command
ENDIF
IF _TC=1                    If error is invalid command (TY)
XQ _ED3,_ED1,1              Skip invalid command
ENDIF
EN                          End of command error routine
```

Example - Communication Interrupt

A DMC-4210 is used to move the A axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an auxiliary port terminal.

```
#BEGIN                                  Label for beginning of program
CC 9600,0,1,0                           Setup communication configuration for auxiliary serial port
CI 2                                    Setup communication interrupt for auxiliary serial port
MG {P2}"Type 0 to stop motion"          Message out of auxiliary port
MG {P2}"Type 1 to pause motion"         Message out of auxiliary port
MG {P2}"Type 2 to resume motion"        Message out of auxiliary port
rate=2000                               Variable to remember speed
SPA=rate                                Set speed of A axis motion
#LOOP                                   Label for Loop
PAA=10000                               Move to absolute position 10000
BGA                                     Begin Motion on A axis
AMA                                     Wait for motion to be complete
PAA=0                                   Move to absolute position 0
BGA                                     Begin Motion on A axis
AMA                                     Wait for motion to be complete
JP #LOOP                                Continually loop to make back and forth motion
EN                                      End main program
#COMINT                                   Interrupt Routine
JP #STOP,P2CH="0"                         Check for S (stop motion)
JP #PAUSE,P2CH="1"                        Check for P (pause motion)
JP #RESUME,P2CH="2"                       Check for R (resume motion)
EN1,1                                     Do nothing
#STOP                                     Routine for stopping motion
STA;ZS;EN                                 Stop motion on A axis; Zero program stack; End Program
#PAUSE                                    Routine for pausing motion
rate=_SPA                                 Save current speed setting of A axis motion
SPA=0                                     Set speed of A axis to zero (allows for pause)
EN1,1                                     Re-enable trippoint and communication interrupt
#RESUME                                   Routine for resuming motion
SPA=rate                                  Set speed on A axis to original speed
EN1,1                                     Re-enable trippoint and communication interrupt
```

For additional information, see section on Using Communication Interrupt.

Example – Ethernet Communication Error

This simple program executes in the DMC-42x0 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

```
#LOOP                          Simple program loop
JP#LOOP
EN
#TCPERR                        Ethernet communication error auto routine
MG {P1}_IA4                    Send message to serial port indicating which handle did not
                               receive proper acknowledgment.
RE
```

Example – Amplifier Error

The program below will execute upon the detection of an error from an internal Galil Amplifier. The bits in TA1 will be set for all axes that have an invalid hall state even if BR1 is set for those axes, this is handled with the mask variable shown in the code below.

```
#AMPERR
REM mask out axes that are in brushed mode for _TA1
mask=(_BRH*128)+(_BRG*64)+(_BRF*32)+(_BRE*16)+(_BRD*8)+(_BRC*4)+(_BRB*2)+_BRA
mask=@COM[mask]
mask=((_TA1&mask)&$0000FFFF)
LU0;'turn off auto update of LCD
REM amplifier error status on LCD
MG"A-ER TA0"{L1},_TA0{L2};WT2000
MG"A-ER TA1"{L1},mask{L2};WT2000
MG"A-ER TA2"{L1},_TA2{L2};WT2000
MG"A-ER TA3"{L1},_TA3{L2};WT2000
LU1;'turn on Automatic Axis Update of LCD
WT5000
REM the sum of the amperr bits should be 0 with no amplifier error
er=_TA0+mask+_TA2+_TA3
JP#AMPERR,er0
REM Notify user amperr has cleared
LU0
MG"AMPERR"{L1},"RESOLVED"{L2}
WT3000
LU1
RE
```

JS Subroutine Stack Variables (^a, ^b, ^c, ^d, ^e, ^f, ^g, ^h)

There are 8 variables that may be passed on the subroutine stack when using the JS command. Passing values on the stack is advanced DMC programming, and is recommended for experienced DMC programmers familiar with the concept of passing arguments by value and by reference.

Notes:

1. Passing parameters has no type checking, so it is important to exercise good programming style when passing parameters. See examples below for recommended syntax.
2. Do not use spaces in expressions containing ^.
3. Global variables MUST be assigned prior to any use in subroutines where variables are passed by reference.
4. Please refer to the JS command in the controller's command reference for further important information.

Example: A Simple Adding Function

```
#Add
JS#SUM(1,2,3,4,5,6,7,8)              ;' call subroutine, pass values
MG_JS                                ;' print return value
EN
'
#SUM                                 ;NO(^a,^b,^c,^d,^e,^f,^g,^h) syntax note for use
EN,,(^a+^b+^c+^d+^e+^f+^g+^h)        ;' return sum

:Executed program from program1.dmc
36.0000
```

Example: Variable, and an Important Note about Creating Global Variables

```
#Var
value=5                                 ;'a value to be passed by reference
global=8                                ;'a global variable
JS#SUM(&value,1,2,3,4,5,6,7)            ;'note first arg passed by reference
MG value                                ;'message out value after subroutine.
MG _JS                                  ;'message out returned value
EN
'
#SUM                                    ;NO(* ^a,^b,^c,^d,^e,^f,^g)
^a=^b+^c+^d+^e+^f+^g+^h+global
EN,,^a
'notes:
'do not use spaces when working with ^
'If using global variables, they MUST be created before the subroutine is run
```

Executed program from program2.dmc

```
36.0000
36.0000
```

Example: Working with Arrays

```
#Array
DM speeds[8]
DM other[256]
JS#zeroAry("speeds",0)                  ;'zero out all buckets in speeds[]
JS#zeroAry("other",0)                   ;'zero out all buckers in other[]
EN
'
#zeroAry                                ;NO(array ^a, ^b) zeros array starting at index ^b
^a[^b]=0
^b=^b+1
JP#zeroAry,(^b<^a[-1])                  ;'[-1] returns the length of an array
EN
```

Example: Abstracting Axes

```
#Axes
JS#runMove(0,10000,1000,100000,100000)
MG "Position:",_JS
EN
'
#runMove                ;NO(axis ^a, PR ^b, SP ^c, AC ^d, DC ^e) Profile movement for axis
~a=^a                   ;'~a is global, so use carefully in subroutines
                         'try one variable axis a-h for each thread A-H
PR~a=^b
SP~a=^c
AC~a=^d
DC~a=^e
BG~a
MC~a
EN,,_TP~a
```

Example: Local Scope

```
#Local
JS#POWER(2,2)
MG_JS
JS#POWER(2,16)
MG_JS
JS#POWER(2,-8)
MG_JS
'
#POWER          ;NO(base ^a,exponent^b) Returns base^exponent power. ± integer only
^c=1            ;'unpassed variable space (^c-^h) can be used as local scope variables
IF ^b=0         ;'special case, exponent = 0
 EN,,1
ENDIF
IF ^b<0         ;'special case, exponent < 0, invert result
 ^d=1
 ^b=@ABS[^b]
ELSE
 ^d=0
ENDIF
#PWRHLPR
^c=^c*^a
^b=^b-1
JP#PWRHLPR,^b>0
IF ^d=1         ;'if inversion required
 ^c=(1/^c)
ENDIF
EN,,^c

Executed program from program1.dmc
4.0000
65536.0000
0.0039
```

Example: Recursion

```
'although the stack depth is only 16, Galil DMC code does support recursion
JS#AxsInfo(0)
MG{Z2.0}"Recursed through ",_JS," stacks"
EN
'
#AxsInfo                            ;NO(axis ^a) List info for axes
~h=^a
^b=(^a+$41)*$1000000                ;'convert to Galil String
MG^b{S1}, " Axis: "{N}
MG{F8.0}"Position: ",_TP~h," Error:",_TE~h," Torque:",_TT~h{F1.4}
IF ^a=7                             ;'recursion exit condition
EN,,1
ENDIF
JS#AxsInfo(^a + 1)                  ;'stack up recursion
EN,,_JS+1                           ;' as recursion closes, add up stack depths

Executed program from program1.dmc
A Axis: Position: 00029319 Error: 00001312 Torque: 9.9982
B Axis: Position: -00001612 Error: 00000936 Torque: 1.7253
C Axis: Position: 00001696 Error:-00001076 Torque:-1.9834
D Axis: Position: -00002020 Error: 00001156 Torque: 2.1309
E Axis: Position: 00000700 Error:-00001300 Torque:-2.3963
F Axis: Position: 00000156 Error:-00000792 Torque:-1.4599
G Axis: Position: -00002212 Error: 00001732 Torque: 3.1926
H Axis: Position: 00002665 Error:-00001721 Torque:-3.1723
Recursed through 8 stacks
```

General Program Flow and Timing information

This section will discuss general programming flow and timing information for Galil programming.

REM vs. NO or ' comments

There are 2 ways to add comments to a .dmc program. REM statements or NO/ ' comments. The main difference between the 2 is that REM statements are stripped from the program upon download to the controller and NO or ' comments are left in the program. In most instances the reason for using REM statements instead of NO or ' is to save program memory. The other benefit to using REM commands comes when command execution of a loop, thread or any section of code is critical. Although they do not take much time, NO and ' comments still take time to process. So when command execution time is critical, REM statements should be used. The 2 examples below demonstrate the difference in command execution of a loop containing comments.

The GalilTools software will treat an apostrophe (') comment different from an NO when the compression algorithm is activated upon a program download (line > 80 characters or program memory > 4000 lines). In this case the software will remove all (') comments as part of the compression and it will download all NO comments to the controller.

Note: Actual processing time will vary depending upon number of axes, communication activity, number of threads currently executing etc.

```
#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
NO this comment takes time to process
'this comment takes time to process
i=i+1;'this comment takes time to process
JP#loop,i<1000
MG TIME-t;'display number of samples from initial time reference
EN
```

When executed on a DMC-4220, the output from the above program returned a 116, which indicates that it took 116 samples (TM 1000) to process the commands from 't=TIME' to 'MG TIME-t'. This is about 114ms ±2ms.

Now when the comments inside of the #loop routine are changed into REM statements (a REM statement must always start on a new line), the processing is greatly reduced.

When executed on the same DMC-4220, the output from the program shown below returned a 62, which indicates that it took 62 samples to process the commands from 't=TIME' to 'MG TIME-t'. This is about 60ms ±2ms, and about 50% faster than when the comments where downloaded to the controller.

```
#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
REM this comment is removed upon download and takes no time to process
REM this comment is removed upon download and takes no time to process
i=i+1
REM this comment is removed upon download and takes no time to process
JP#loop,i<1000
MG TIME-t;'display number of samples from initial time reference
EN
```

WT vs AT and coding deterministic loops

The main difference between WT and AT is that WT will hold up execution of the next command for the specified time from the execution of the WT command, AT will hold up execution of the next command for the specified time from the last time reference set with the AT command.

```
#A
AT0;'set initial AT time reference
WT 1000,1;'wait 1000 samples
t1=TIME
AT 4000,1;'wait 4000 samples from last time reference
t2=TIME-t1
REM in the above scenario, t2 will be ~3000 because AT 4000,1 will have
REM paused program execution from the time reference of AT0
REM since the WT 1000,1 took 1000 samples, there was only 3000 samples left
REM of the "4000" samples for AT 4000,1
MG t,t2;'this should output 1000,3000
EN;'End program
```

Where the functionality of the operation of the AT command is very useful is when it is required to have a deterministic loop operating on the controller. These instances range from writing PLC-type scan threads to writing custom control algorithms. The key to having a deterministic loop time is to have a trippoint that will wait a specified time independent of the time it took to execute the loop code. In this definition, the AT command is a perfect fit. The below code is an example of a PLC-type scan thread that runs at a 500ms loop rate. A typical implementation would be to run this code in a separate thread (ex XQ#plcscan,2).

```
REM this code will set output 3 high if
REM inputs 1 and 2 are high, and input 3 is low
REM else output 3 will be low
REM if input 4 is low, output 1 will be high
REM and ouput 3 will be low regardless of the
REM states of inputs 1,2 or 3
#plcscan
AT0;'set initial time reference
#scan
REM mask inputs 1-4
ti=_TI0&$F
REM variables for bit 1 and bit 3
b1=0;b3=0
REM if input 4 is high set bit 1 and clear bit 3
REM ti&8 - gets 4th bit, if 4th bit is high result = 8
IF ti&8=8;b1=1;ELSE
REM ti&7 get lower 3 bits, if 011 then result = 3
IF ti&7=3;b3=1;ENDIF;ENDIF
REM set output bits 1 and 3 accordingly
REM set outputs at the end for a PLC scan
OB1,b1;OB3,b3
REM wait 500ms (for 500 samples use AT-500,1)
REM the '-' will reset the time reference
AT-500
JP#scan
```

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-42x0 provides the use of the following mathematical operators:

Operator	Function
+	Addition
−	Subtraction
*	Multiplication
/	Division
%	Modulus
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
()	Parenthesis

Mathematical operations are executed from left to right. Calculations within parentheses have precedence.

Examples:

`speed = 7.5*V1/2`	The variable, speed, is equal to 7.5 multiplied by V1 and divided by 2
`count = count+2`	The variable, count, is equal to the current value plus 2.
`result =_TPX-(@COS[45]*40)`	Puts the position of X - 28.28 in result. 40 * cosine of 45° is 28.28
`temp = @IN[1]&@IN[2]`	temp is equal to 1 only if Input 1 and Input 2 are high

Mathematical Operation Precision and Range

The controller stores non-integers in a fixed point representation (not floating point). Numbers are stored as 4 bytes of integer and 2 bytes of fraction within the range of ± 2,147,483,647.9999. The smallest number representable (and thus the precision) is 1/65536 or approximately 0.000015.

Example:

Using basic mathematics it is known that 1.4*(80,000) = 112,000. However, using a basic terminal, a DMC controller would calculate the following:

```
:var= 1.4*80000;'       Storing the result of 1.4*80000 in var
:MG var;'               Prints variable "var" to screen
111999.5117
:
```

The reason for this error relies in the precision of the controller. 1.4 must be stored to the nearest multiple of 1/65536, which is 91750/65536 = 1.3999. Thus, (91750/65536)*80000 = 111999.5117 and reveals the source of the error.

By ignoring decimals and multiplying by integers first (since they carry no error), and then adding the decimal back in by dividing by a factor of 10 will allow the user to avoid any errors caused by the limitations of precision of the controller. Continuing from the example above:

```
:var= 14*80000;'        Ignore decimals
:MG var;'               Print result
1120000.0000
:var= var/10;'          Divide by 10 to add in decimal
:MG var;'               Print correct result
112000.0000
:
```

Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid DMC-42x0 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value which is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

```
#TEST                              Begin main program
IN "ENTER",len{S6}                 Input character string of up to 6 characters into variable 'len'
Flen=@FRAC[len]                    Define variable 'Flen' as fractional part of variable 'len'
Flen=$10000*Flen                   Shift Flen by 32 bits (IE - convert fraction, Flen, to integer)
len1=(Flen&$00FF)                  Mask top byte of Flen and set this value to variable 'len1'
len2=(Flen&$FF00)/$100             Let variable, 'len2' = top byte of Flen
len3=len&$000000FF                 Let variable, 'len3' = bottom byte of len
len4=(len&$0000FF00)/$100          Let variable, 'len4' = second byte of len
len5=(len&$00FF0000)/$10000        Let variable, 'len5' = third byte of len
len6=(len&$FF000000)/$1000000      Let variable, 'len6' = fourth byte of len
MG len6 {S4}                       Display 'len6' as string message of up to 4 chars
MG len5 {S4}                       Display 'len5' as string message of up to 4 chars
MG len4 {S4}                       Display 'len4' as string message of up to 4 chars
MG len3 {S4}                       Display 'len3' as string message of up to 4 chars
MG len2 {S4}                       Display 'len2' as string message of up to 4 chars
MG len1 {S4}                       Display 'len1' as string message of up to 4 chars
EN
```

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding '$'). For more information, see section Sending Messages.

To illustrate further, if the user types in the string "TESTME" at the input prompt, the controller will respond with the following:

```
T                                  Response from command MG len6 {S4}
E                                  Response from command MG len5 {S4}
S                                  Response from command MG len4 {S4}
T                                  Response from command MG len3 {S4}
M                                  Response from command MG len2 {S4}
E                                  Response from command MG len1 {S4}
```

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@COS[n]	Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@TAN[n]	Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@ASIN*[n]	Arc Sine of n, between -90° and +90°. Angle resolution in 1/64000 degrees.
@ACOS*[n]	Arc Cosine of n, between 0 and 180°. Angle resolution in 1/64000 degrees.
@ATAN*[n]	Arc Tangent of n, between -90° and +90°. Angle resolution in 1/64000 degrees
@COM[n]	1's Complement of n
@ABS[n]	Absolute value of n
@FRAC[n]	Fraction portion of n
@INT[n]	Integer portion of n
@RND[n]	Round of n (Rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of n (Accuracy is ±.004)
@IN[n]	Return digital input at general input n (where n starts at 1)
@OUT[n]	Return digital output at general output n (where n starts at 1)
@AN[n]	Return analog input at general analog in n (where n starts at 1)

*Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

Examples:

```
v1=@ABS[V7]        The variable, v1, is equal to the absolute value of variable v7.
v2=5*@SIN[pos]     The variable, v2, is equal to five times the sine of the variable, pos.
v3=@IN[1]          The variable, v3, is equal to the digital value of input 1.
v4=2*(5+@AN[5])    The variable, v4, is equal to the value of analog input 5 plus 5, then multiplied by 2.
```

Variables

For applications that require a parameter that is variable, the DMC-42x0 provides 510 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Example:

```
posx=5000          Assigns the value of 5000 to the variable posx
PR posx            Assigns variable posx to PR command
JG rpmY*70         Assigns variable rpmY multiplied by 70 to JG command.
```

Programmable Variables

The DMC-42x0 allows the user to create up to 510 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character; however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-42x0 instructions. For example, PR is not a good choice for a variable name.

Note: It is generally a good idea to use lower-case variable names so there is no confusion between Galil commands and variable names.

Examples of valid and invalid variable names are:

Valid Variable Names

```
posx
pos1
speedZ
```

Invalid Variable Names

```
RealLongName   ; 'Cannot have more than 8 characters
123            ; 'Cannot begin variable name with a number
speed Z        ; 'Cannot have spaces in the name
```

Assigning Values to Variables:

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings. The range for numeric variable values is 4 bytes of integer (231) followed by two bytes of fraction (±2,147,483,647.9999).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid DMC-42x0 function can be used to assign a value to a variable. For example, v1=@ABS[v2] or v2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters which must be in quotation.

Examples:

```
posX=_TPX          Assigns returned value from TPX command to variable posx.
speed=5.75         Assigns value 5.75 to variable speed
input=@IN[2]       Assigns logical value of input 2 to variable input
v2=v1+v3*v4        Assigns the value of v1 plus v3 times v4 to the variable v2.
var="CAT"          Assign the string, CAT, to var
MG var{S3}         Displays the variable var – (CAT)
```

Assigning Variable Values to Controller Parameters

Variable values may be assigned to controller parameters such as SP or PR.

```
PR v1              Assign v1 to PR command
```

SP vS*2000	Assign vS*2000 to SP command

Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, variable=. For example, v1= , returns the value of the variable v1.

Example - Using Variables for Joystick

The example below reads the voltage of an X-Y joystick and assigns it to variables vX and vY to drive the motors at proportional velocities, where:

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

```
#JOYSTIK          Label
JG 0,0            Set in Jog mode
BGXY              Begin Motion
AT0               Set AT time reference
#LOOP             Loop
vX=@AN[1]*20000   Read joystick X
vY=@AN[2]*20000   Read joystick Y
JG vX,vY          Jog at variable vX,vY
AT-4              Wait 4ms from last time reference, creates a deterministic loop time
JP#LOOP           Repeat
EN                End
```

Operands

Operands allow motion or status parameters of the DMC-42x0 to be incorporated into programmable variables and expressions. Most DMC commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-42x0 command. The command reference indicates which commands have an associated operand.

Status commands such as Tell Position return actual values, whereas action commands such as KP or SP return the values in the DMC-42x0 registers. The axis designation is required following the command.

Examples of Internal Variables:

```
posX=_TPX          Assigns value from Tell Position X to the variable posX.
deriv=_KDZ*2       Assigns value from KDZ multiplied by two to variable, deriv.
JP #LOOP,_TEX>5    Jump to #LOOP if the position error of X is greater than 5
JP #ERROR,_TC=1    Jump to #ERROR if the error code equals 1.
```

Operands can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _KDX=2 is invalid.

Special Operands (Keywords)

The DMC-42x0 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-42x0 commands.

Keyword	Function
_BGn	*Returns a 1 if motion on axis 'n' is complete, otherwise returns 0.
_BN	*Returns serial # of the board.
_DA	*Returns the number of arrays available
_DL	*Returns the number of available labels for programming
_DM	*Returns the available array memory
_HMn	*Returns status of Home Switch (equals 0 or 1)
_LFn	Returns status of Forward Limit switch input of axis 'n' (equals 0 or 1)
_LRX	Returns status of Reverse Limit switch input of axis 'n' (equals 0 or 1)
_UL	*Returns the number of available variables
TIME	Free-Running Real Time Clock (off by 2.4% - Resets with power-on). Note: TIME does not use an underscore character (_) as other keywords.

* - These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Reference.

Examples of Keywords:

```
v1=_LFX     Assign V1 the logical state of the Forward Limit Switch on the X-axis
v3=TIME     Assign V3 the current value of the time clock
v4=_HMW     Assign V4 the logical state of the Home input on the W-axis
```

Arrays

For storing and collecting numerical data, the DMC-42x0 provides array space for 24000 elements. The arrays are one dimensional and up to 30 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (2^{31}) followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an alphabetic character. The number of entries in the defined array is enclosed in [ ].

Example:

```
DM posx[7]          Defines an array names 'posx' with seven entries
DM speed[100]       Defines an array named speed with 100 entries
DA posx[]           Frees array space
```

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the 'posx' array (defined with the DM command, DM posx[7]) would be specified as posx[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

Examples:

`DM speed[10]`	Dimension speed Array
`speed[0]=7650.2`	Assigns the first element of the array, 'speed' the value 7650.2
`speed[0]=`	Returns array element value
`posx[10]=_TPX`	Assigns the 10th element of the array 'posx' the returned value from the tell position command.
`con[1]=@COS[pos]*2`	Assigns the second element of the array 'con' the cosine of the variable POS multiplied by 2.
`timer[0]=TIME`	Assigns the first element of the array timer the returned value of the TIME keyword.

Using a Variable to Address Array Elements

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

Example:

`#A`	Begin Program
`count=0;DM pos[10]`	Initialize counter and define array
`#LOOP`	Begin loop
`WT 10`	Wait 10 msec
`pos[count]=_TPX`	Record position into array element
`pos[count]=`	Report position
`count=count+1`	Increment counter
`JP #LOOP,count<10`	Loop until 10 elements have been stored
`EN`	End Program

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named 'pos'. The variable, 'count', is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

Uploading and Downloading Arrays to On Board Memory

The GalilTools software is recommended for downloading and uploading array data from the controller. The GalilTools Communication library also provides function calls for downloading and uploading array data from the controller to/from a buffer or a file.

Arrays may also be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

start is the first element of array (default=0)

end is the last element of array (default=last element)

delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The DMC-42x0 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in eight arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

Command Summary - Automatic Data Capture

Command	Description
RA n[ ],m[ ],o[ ],p[ ]	Selects up to eight arrays for data capture. The arrays must be defined with the DM command.
RD type1,type2,type3,type4	Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see table below). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2^n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording.
RC?	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Data Types for Recording:

Data type	Description
TIME	Controller time as reported by the TIME command
_AFn	Analog input (n=X,Y,Z,W,E,F,G,H, for AN inputs 1-8)
_DEX	2nd encoder position (dual encoder)
_NOX	Status bits
_OP	Output
_RLX	Latched position
_RPX	Commanded position
_SCX	Stop code
_TEX	Position error
_TI	Inputs
_TPX	Encoder position
_TSX	Switches (only bit 0-4 valid)
_TTX	Torque (reports digital value ±32544)

Note: X may be replaced by Y,Z or W for capturing data on other axes.

Operand Summary - Automatic Data Capture

_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress
_RD	Returns address of next array element.

Example - Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

```
#RECORD                              Begin program
DM XPOS[300],YPOS[300]               Define X,Y position arrays
DM XERR[300],YERR[300]               Define X,Y error arrays
RA XPOS[],XERR[],YPOS[],YERR[]       Select arrays for capture
RD _TPX,_TEX,_TPY,_TEY               Select data types
PR 10000,20000                       Specify move distance
RC1                                  Start recording now, at rate of 2 msec
BG XY                                Begin motion
#A;JP #A,_RC=1                       Loop until done
MG "DONE"                            Print message
EN                                   End program
#PLAY                                Play back
N=0                                  Initial Counter
JP# DONE,N>300                       Exit if done
N=                                   Print Counter
X POS[N]=                            Print X position
Y POS[N]=                            Print Y position
XERR[N]=                             Print X error
YERR[N]=                             Print Y error
N=N+1                                Increment Counter
#DONE                                Done
EN                                   End Program
```

De-allocating Array Space

Array space may be de-allocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Input of Data (Numeric and String)

Sending Data from a Host

The DMC unit can accept ASCII strings from a host. This is the most common way to send data to the controller such as setting variables to numbers or strings. Any variable can be stored in a string format up to 6 characters by simply specifying defining that variable to the string value with quotes, for example:

```
varS = “STRING”
```

Will assign the variable 'varS' to a string value of “STRING”.

To assign a variable a numerical value, the direct number is used, for example:

```
varN = 123456
```

Will assign the variable 'varN' to a number of 123,456.

All variables on the DMC controller are stored with 4 bytes of integer and 2 bytes of fractional data.

Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the auxiliary RS-232 port. In this mode, the DMC-42x0 provides a buffer for receiving characters. This mode may only be used when executing an applications program.

The Operator Data Entry Mode may be specified for Port 2 only. This mode may be exited with the \ or <escape> key.

NOTE: Operator Data Entry Mode cannot be used for high rate data transfer.

Set the third field of the CC command to one to set the Operator Data Entry Mode.

To capture and decode characters in the Operator Data Mode, the DMC-42x0 provides special the following keywords:

Keyword	Function
P2CH	Contains the last character received
P2ST	Contains the received string
P2NM	Contains the received number
P2CD	Contains the status code: -1 mode disabled 0 nothing received 1 received character, but not <enter> 2 received string, not a number 3 received number

NOTE: The value of P2CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data and they may also be used in conditional statements with logical operators.

Example

Instruction	Interpretation
`JP #LOOP,P2CD< >3`	Checks to see if status code is 3 (number received)
`JP #P,P2CH="V"`	Checks if last character received was a V
`PR P2NM`	Assigns received number to position
`JS #XAXIS,P2ST="X"`	Checks to see if received string is X

Using Communication Interrupt

The DMC-42x0 provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is CI n:

n = 0	Don't interrupt Port 2
n = 1	Interrupt on <enter> Port 2
n = 2	Interrupt on any character Port 2
n = -1	Clear any characters in buffer

The #COMINT label is used for the communication interrupt. For example, the DMC-42x0 can be configured to interrupt on any character received on Port 2. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

Example

A DMC-42x0 is used to jog the A and B axis. This program automatically begins upon power-up and allows the user to input values from the main serial port terminal. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An S stops motion on both axes.

Instruction	Interpretation
`#AUTO`	Label for Auto Execute
`speedA=10000`	Initial A speed
`speedB=10000`	Initial B speed
`CI 2`	Set Port 2 for Character Interrupt
`JG speedA, speedB`	Specify jog mode speed for A and B axis
`BGXY`	Begin motion
`#PRINT`	Routine to print message to terminal
`MG{P2}"TO CHANGE SPEEDS"`	Print message
`MG{P2}"TYPE A OR B"`	
`MG{P2}"TYPE S TO STOP"`	
`#JOGLOOP`	Loop to change Jog speeds
`JG speedA, speedB`	Set new jog speed
`JP #JOGLOOP`	
`EN`	End of main program
`#COMINT`	Interrupt routine
`JP #A,P2CH="A"`	Check for A
`JP #B,P2CH="B"`	Check for B
`JP #C,P2CH="S"`	Check for S
`ZS1;CI2;JP#JOGLOOP`	Jump if not X,Y,S
`#A;JS#NUM`	
`speedX=val`	New X speed
`ZS1;CI2;JP#PRINT`	Jump to Print
`#B;JS#NUM`	
`speedY=val`	New Y speed
`ZS1;CI2;JP#PRINT`	Jump to Print
`#C;ST;AMX;CI-1`	Stop motion on S
`MG{^8}, "THE END"`	
`ZS;EN,1`	End-Re-enable interrupt
`#NUM`	Routine for entering new jog speed
`MG "ENTER",P2CH{S},"AXIS SPEED" {N}`	Prompt for value
`#NUMLOOP; CI-1`	Check for enter

```
#NMLP                          Routine to check input from terminal
JP #NMLP,P2CD<2                Jump to error if string
JP #ERROR,P2CD=2               Read value
val=P2NM
EN                             End subroutine
#ERROR;CI-1                    Error Routine
MG "INVALID-TRY AGAIN"         Error message
JP #NMLP
EN                             End
```

Inputting String Variables

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter A,B or C", V{S} specifies a string variable to be input.

The DMC-42x0, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see section Bit-wise Operators.

Output of Data (Numeric and String)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", result
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Analog input is", @AN[1]
MG "The Position of A is", _TPA
```

Specifying the Port for Messages:

The port can be specified with the specifier, {P1} for the main serial port {P2} for auxiliary serial port, or {En} for the Ethernet port.

```
MG {P2} "Hello World"          Sends message to Auxiliary Port
```

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example:

MG "The Final Value is", result {F5.2}

If the value of the variable result is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable result is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
 #A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.0} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

The Speed is 50000 counts/sec

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

MG {^07} {^255}

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions

Function	Description
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the left of the decimal point and m digits to the right
{P1}, {P2} or {En}	Send message to Main Serial Port, Auxiliary Serial Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= or array[x]=. For example, v1= returns the value of v1.

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
DM posA[7]	Define Array posA with 7 entries
PR 1000	Position Command
BGX	Begin
AMX	After Motion
v1=_TPA	Assign Variable v1
posA[1]=_TPA	Assign the first entry
v1=	Print v1

Interrogation Commands

The DMC-42x0 has a set of commands that directly interrogate the controller. When these command are entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), and Leading Zeros (LZ) command. For a complete description of interrogation commands, see Chapter 5.

Using the PF Command to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by theses interrogation commands:

```
BL ?                LE ?
DE ?                PA ?
DP ?                PR ?
EM ?                TN ?
FL ?                VE ?
IP ?                TE
TP
```

The numeric values may be formatted in decimal or hexadecimal with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

```
PF m.n
```

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example

Instruction	Interpretation
:DP21	Define position
:TPA	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPA	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPA	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places
:TPA	Tell Position
99	Returns 99 if position greater than 99

Adding Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be added by the use of the command, LZ. The LZ command is set to a default of 1.

```
LZ0                                  Disables the LZ function
TP                                   Tell Position Interrogation Command
-0000000009, 0000000005              Response (With Leading Zeros)
LZ1                                  Enables the LZ function
TP                                   Tell Position Interrogation Command
-9, 5                                Response (Without Leading Zeros)
```

Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

```
TP {F2.2}                            Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00          Response from Interrogation Command
TP {$4.2}                            Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00   Response from Interrogation Command
```

Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

VF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Response - Default format
VF2.2	Change format
v1=	Return v1
:10.00	Response - New format
VF-2.2	Specify hex format
v1=	Return v1
$0A.00	Response - Hex value
VF1	Change format
v1=	Return v1
:9	Response - Overflow

Local Formatting of Variables

PF and VF commands are global format commands that affect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Default Format
v1={F4.2}	Specify local format
:0010.00	New format
v1={$4.2}	Specify hex format
:$000A.00	Hex value
v1="ALPHA"	Assign string "ALPHA" to v1
v1={S4}	Specify string format first 4 characters
:ALPH	

The local format is also used with the MG command.

Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The DMC-42x0 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

Instruction	Interpretation
#RUN	Label
MG "ENTER # OF REVOLUTIONS";n1=-1	Prompt for revs
#rev;JP#rev,n1=-1	Wait until user enters new value for n1
PR n1*2000	Convert to counts
MG "ENTER SPEED IN RPM";s1=-1	Prompt for RPMs
#spd;JP#spd,s1=-1	Wait for user to enter new value for s1
SP s1*2000/60	Convert to counts/sec
MG "ENTER ACCEL IN RAD/SEC2";a1=-1	Prompt for ACCEL
#acc;JP#acc,a1=-1	Wait for user to enter new value for a1
AC a1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

Hardware I/O

Digital Outputs

The DMC-42x0 has an 8-bit uncommitted output port and an additional 32 I/O which may be configured as inputs or outputs with the CO command for controlling external events. The DMC-4250 through DMC-4280 has an additional 8 outputs. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB (Clear Bit), or OB (define output bit).

Example- Set Bit and Clear Bit

Instruction	Interpretation
SB6	Sets bit 6 of output port
CB4	Clears bit 4 of output port

Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Interpretation
`OB1, POS`	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
`OB 2, @IN [1]`	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
`OB 3, @IN [1]&@IN [2]`	Set Output 3 only if Input 1 and Input 2 are high.
`OB 4, COUNT [1]`	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 16-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 16-bit output port, where bit 0 is output 1, bit1 is output2 and so on. A 1 designates that the output is on.

Example- Output Port

Instruction	Interpretation
`OP6`	Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$)
`OP0`	Clears all bits of output port to zero
`OP 255`	Sets all bits of output port to one. ($2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$)

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

Example - Turn on output after move

Instruction	Interpretation
`#OUTPUT`	Label
`PR 2000`	Position Command
`BG`	Begin
`AM`	After move
`SB1`	Set Output 1
`WT 1000`	Wait 1000 msec
`CB1`	Clear Output 1
`EN`	End

Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n, where n is a number 1 through 48.

Example - Using Inputs to control program flow

Instruction	Interpretation
`JP #A,@IN[1]=0`	Jump to A if input 1 is low
`JP #B,@IN[2]=1`	Jump to B if input 2 is high
`AI 7`	Wait until input 7 is high
`AI -6`	Wait until input 6 is low

Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor A must stop turning.

Solution: Connect panel switch to input 1 of DMC-42x0. High on input 1 means switch is in on position.

Instruction	Interpretation
`#S;JG 4000`	Set speed
`AI 1;BGA`	Begin after input 1 goes high
`AI -1;STA`	Stop after input 1 goes low
`AMA;JP #S`	After motion, repeat
`EN`	

The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. For each axis, the controller has one auxiliary encoder and each auxiliary encoder consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-96.

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between ± 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 5 volts if the signal is a 0 - 12 volt logic).

Example:

A DMC-4210 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

NOTE: The auxiliary encoder inputs are not available for any axis that is configured for stepper motor.

Input Interrupt Function

The DMC-42x0 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. For example, II,,5 enables inputs 1 and 3.

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

IMPORTANT	Use the RI command (not EN) to return from the #ININT subroutine.

Example - Input Interrupt

Instruction	Interpretation
`#A`	Label #A
`II 1`	Enable input 1 for interrupt function
`JG 30000,-20000`	Set speeds on A and B axes
`BG AB`	Begin motion on A and B axes
`#B`	Label #B
`TP AB`	Report A and B axes positions
`WT 1000`	Wait 1000 milliseconds
`JP #B`	Jump to #B
`EN`	End of program
`#ININT`	Interrupt subroutine
`MG "Interrupt has occurred"`	Displays the message
`ST AB`	Stops motion on A and B axes
`#LOOP;JP #LOOP,@IN[1]=0`	Loop until Interrupt cleared
`JG 15000,10000`	Specify new speeds
`WT 300`	Wait 300 milliseconds
`BG AB`	Begin motion on A and B axes
`RI`	Return from Interrupt subroutine

Jumping back to main program with #ININT

To jump back to the main program using the JP command, the RI command must be issued in a subroutine and then the ZS command must be issued prior to the JP command. See Application Note # 2418 for more information.

http://www.galilmc.com/support/appnotes/optima/note2418.pdf

Analog Inputs

The DMC-42x0 provides eight analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit ADC is available as an option). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command A to move to that point.

Instruction	Interpretation
`#POINTS`	Label
`SP 7000`	Speed
`AC 80000;DC 80000`	Acceleration
`#LOOP`	
`VP=@AN[1]*1000`	Read and analog input, compute position
`PA VP`	Command position
`BGA`	Start motion
`AMA`	After completion
`JP #LOOP`	Repeat
`EN`	End

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

Instruction	Interpretation
#CONT	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#LOOP	
vp=@AN[1]*1000	Compute desired position
ve=vp-_TPA	Find position error
vel=ve*20	Compute velocity
JG vel	Change velocity
JP #LOOP	Change velocity
EN	End

Example – Low Pass Digital Filter for the Analog inputs

Because the analog inputs on the Galil controller can be used to close a position loop, they have a very high bandwidth and will therefor read noise that comes in on the analog input. Often when an analog input is used in a motion control system, but not for closed loop control, the higher bandwidth is not required. In this case a simple digital filter may be applied to the analog input, and the output of the filter can be used for in the motion control application. This example shows how to apply a simple single pole low-pass digital filter to an analog input. This code is commonly run in a separate thread (XQ#filt,1 – example of executing in thread 1).

```
#filt
REM an1 = filtered output.  Use this instead of @AN[1]
an1=@AN[1];'set initial value
REM k1+k2=1 this condition must be met
REM use division of m/2^n for elimination of round off
REM increase k1 = less filtering
REM increase k2 = more filtering
k1=32/64;k2=32/64
AT0;'set initial time reference
#loop
REM calculate filtered output and then way 2 samples from last
REM time reference (last AT-2,1 or AT0)
an1=(k1*@AN[1])+(k2*an1);AT-2,1
JP#loop
```

Extended I/O of the DMC-42x0 Controller

The DMC-42x0 controller offers 32 extended I/O points which can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through 1 44 pin high density connector.

Configuring the I/O of the DMC-42x0

The 32 extended I/O points of the DMC-42x0 series controller can be configured in blocks of 8. The extended I/O is denoted as blocks 2-5 or bits 17-48.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

where n is a decimal value which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

The least significant bit represents block 2 and the most significant bit represents block 5. The decimal value can be calculated by the following formula. n = n2 + 2*n3 + 4*n4 + 5*n5 where nx represents the block. If the nx value is a one, then the block of 8 I/O points is to be configured as an output. If the nx value is a zero, then the block of 8 I/O points will be configured as an input. For example, if block 4 and 5 is to be configured as an output, CO 12 is issued.

8-Bit I/O Block	Block	Binary Representation	Decimal Value for Block
17-24	2	2^0	1
25-32	3	2^1	2
33-40	4	2^2	4
41-48	5	2^3	8

The simplest method for determining n:

Step 1. Determine which 8-bit I/O blocks to be configured as outputs.
Step 2. From the table, determine the decimal value for each I/O block to be set as an output.
Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 2 and 3 are to be outputs, then n is 3 and the command, CO3, should be issued.

NOTE: This calculation is identical to the formula: n = n2 + 2*n3 + 4*n4 + 5*n5 where nx represents the block.

Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the non-volatile flash memory with the BN command. If no value has been set, the default of CO 0 is used (all blocks are inputs).

Accessing Extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=1 through 8 and 17 through 48). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 48).

4080

For 5-8 axis controllers, each I/O point may be defined with the SBn and CBn commands (where n=1 through 48).

The command, OP, may also be used to set output bits, specified as blocks of data. The OP command accepts 3 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, block 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits (values from 0 to 65535). Arguments which are given for I/O points which are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Blocks	Bits	Description
m	0	1-8	General Outputs
a	2,3	17-32	Extended I/O
b	4,5	33-48	Extended I/O

For example, if block 8 is configured as an output, the following command may be issued:

OP 7,,7

This command will set bits 1,2,3 (block 0) and bits 33,34,35 (block 4) to 1. Bits 4 through 8 and bits 36 through 48 will be set to 0. All other bits are unaffected.

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=0,2,3 or 4). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 48). If the following command is issued;

- Individual bits can be queried using the @IN[n] function (where n=1 through 48).

MG @IN[17]

the controller will return the state of the least significant bit of block 2 (assuming block 2 is configured as an input).

Example Applications

Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

$$4000/2\pi = 637 \text{ count/inch}$$

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Figure 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

INSTRUCTION	FUNCTION
`#A`	Label
`AI1`	Wait for input 1
`PR 6370`	Distance
`SP 3185`	Speed
`BGX`	Start Motion
`AMX`	After motion is complete
`SB1`	Set output bit 1
`WT 20`	Wait 20 ms
`CB1`	Clear output bit 1
`WT 80`	Wait 80 ms
`JP #A`	Repeat the process

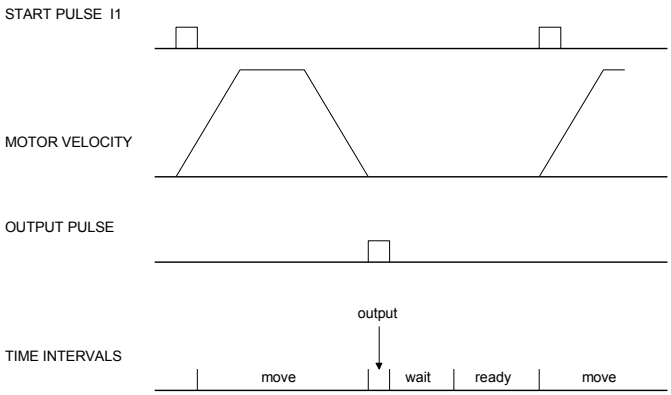


Figure 7.1: Motor Velocity and the Associated Input/Output signals

X-Y Table Controller

An X-Y-Z system must cut the pattern shown in Figure 7.2. The X-Y table moves the plate while the Z-axis raises and lowers the cutting tool.

The solid curves in Figure 7.2 indicate sections where cutting takes place. Those must be performed at a feed rate of 1 inch per second. The dashed line corresponds to non-cutting moves and should be performed at 5 inch per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the Z-axis raised. An X-Y motion to point B is followed by lowering the Z-axis and performing a cut along the circle. Once the circular motion is completed, the Z-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts

and the speeds of

1 in/sec = 40,000 count/sec
5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

0.1g = 38.6 in/s2 = 1,544,000 count/s^2

Note that the circular path has a radius of 2" or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

CR 80000,270,-360

Further assume that the Z must move 2" at a linear speed of 2" per second. The required motion is performed by the following instructions:

INSTRUCTION	FUNCTION
#A	Label
VM XY	Circular interpolation for XY
VP 160000,160000	Positions
VE	End Vector Motion
VS 200000	Vector Speed
VA 1544000	Vector Acceleration
BGS	Start Motion
AMS	When motion is complete
PR,,-80000	Move Z down
SP,,80000	Z speed
BGZ	Start Z motion
AMZ	Wait for completion of Z motion
CR 80000,270,-360	Circle
VE	
VS 40000	Feed rate
BGS	Start circular move
AMS	Wait for completion
PR,,80000	Move Z up
BGZ	Start Z move
AMZ	Wait for Z completion
PR -21600	Move X
SP 20000	Speed X
BGX	Start X
AMX	Wait for X completion
PR,,-80000	Lower Z
BGZ	
AMZ	
CR 80000,270,-360	Z second circle move
VE	

```
VS 40000
BGS
AMS
PR,,80000              Raise Z
BGZ
AMZ
VP -37600,-16000       Return XY to start
VE
VS 200000
BGS
AMS
EN
```

Figure 7.2: Motor Velocity and the Associated Input/Output signals

Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as:

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

Instruction

```
#A
JG0
BGX
#B
VIN=@AN[1]
VEL=VIN*20000
JG VEL
JP #B
EN
```

Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

INSTRUCTION	FUNCTION
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGX	Start
#B	
VIN=@AN[1]	Read analog input
V2=V1*V3	Compute the desired position
V4=V2-_TPX-_TEX	Find the following error
V5=V4*20	Compute a proportional speed
JG V5	Change the speed
JP #B	Repeat the process
EN	End

Backlash Compensation by Sampled Dual-Loop

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a lead screw. Such a lead screw has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of ±2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position

is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below ±2 counts. Often, this is performed in one correction cycle.

Example:

INSTRUCTION	FUNCTION
#A	Label
DP0	Define starting positions as zero
LINPOS=0	
PR 1000	Required distance
BGX	Start motion
#B	
AMX	Wait for completion
WT 50	Wait 50 msec
LINPOS = _DEX	Read linear position
ERR=1000-LINPOS-_TEX	Find the correction
JP #C,@ABS[ERR]<2	Exit if error is small
PR ERR	Command correction
BGX	
JP #B	Repeat the process
#C	
EN	

Using the DMC Editor to Enter Programs

The GalilTools software package provides an editor and utilities that allow the upload and download of DMC programs to the motion controller.

Application programs for the DMC-42x0 may also be created and edited locally using the DMC-42x0.

The DMC-42x0 provides a line Editor for entering and modifying programs. The Edit mode is entered with the ED instruction. (Note: The ED command can only be given when the controller is in the non-edit mode, which is signified by a colon prompt).

In the Edit Mode, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the last program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

ED	Puts Editor at end of last program
:ED 5	Puts Editor at line 5
:ED #BEGIN	Puts Editor at label #BEGIN

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed 80 characters per line.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

Edit Mode Commands

<RETURN>

Typing the return key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P

The <cntrl>P command moves the editor to the previous line.

<cntrl>I

The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labeled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D

The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q

The <cntrl>Q quits the editor mode. In response, the DMC-42x0 will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no operand follows the LS command, the entire program will be listed. The user can start listing at a specific line or label using the operand n. A command and new line number or label following the start listing operand specifies the location at which listing is to stop.

Example:

Instruction	Interpretation
`:LS`	List entire program
`:LS 5`	Begin listing at line 5
`:LS 5,9`	List lines 5 thru 9
`:LS #A,9`	List line label #A thru line 9
`:LS #A, #A +5`	List line label #A and additional 5 lines

Chapter 8 Hardware & Software Protection

Introduction

The DMC-42x0 provides several hardware and software features to check for error conditions and to inhibit the motor on error. These features help protect the various system components from damage.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the DMC-42x0 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-42x0. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-42x0 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

Output Protection Lines

Amp Enable

This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has separate amplifier enable lines. This signal also goes low when the watch-dog timer is activated, or upon reset.

Note: The standard configuration of the AEN signal is high amplifier enable (HAEN). Both the polarity and the amplitude can be changed. To make these changes, see section entitled Amplifier Circuit in Chapter 3.

Error Output

The error output is a TTL signal which indicates an error condition in the controller. This signal is available on the interconnect module as ERR. When the error signal is low, this indicates an error condition and the Error Light on the controller will be illuminated. For details on the reasons why the error output would be active see Error Light (Red LED) in Chapter 9.

Input Protection Lines

General Abort

A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

The Abort input by default will also halt program execution; this can be changed by changing the 5th field of the CN command. See the CN command in the command reference for more information.

Selective Abort

The controller can be configured to provide an individual abort for each axis. Activation of the selective abort signal will act the same as the Abort Input but only on the specific axis. To configure the controller for selective abort, issue the command CN,,,1. This configures the inputs 5,6,7,8,13,14,15,16 to act as selective aborts for axes A,B,C,D,E,F,G,H respectively.

ELO (Electronic Lock Out)

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see in the Appendices.

Forward Limit Switch

Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

Reverse Limit Switch

Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

Software Protection

The DMC-42x0 provides a programmable error limit as well as encoder failure detection. It is recommended that both the position error and encoder failure detection be used when running servo motors with the DMC-42x0. Along with position error and encoder failure detection, then DMC-42x0 has the ability to have programmable software limit.

Position Error

The error limit can be set for any number between 0 and 2147483647 using the ER n command. The default value for ER is 16384.

Example:

`ER 200,300,400,500`	Set X-axis error limit for 200, Y-axis error limit to 300, Z-axis error limit to 400 counts, W-axis error limit to 500 counts
`ER,1,,10`	Set Y-axis error limit to 1 count, set W-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the controller will generate several signals to warn the host system of the error condition. These signals include:

Signal or Function	State if Error Occurs
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on
OE Function	Shuts motor off if OE1 or OE3
AEN Output Line	Switches to Motor Off state

The Jump on Condition statement is useful for branching on a given error within a program. The position error of X,Y,Z and W can be monitored during execution using the TE command.

Encoder Failure detection

The encoder failure detection on the controller operates based upon two factors that are user settable, a threshold of motor command output (OV), a time above that threshold (OT) in which there is no more than 4 counts of change on the encoder input for that axis. The encoder failure detection is activated with the OA command. When an encoder failure is detected and OA is set to 1 for that axis, the same conditions will occur as a position error.

Conditions for proper operation of Encoder Failure detection

- The axis must have a non-zero KI setting order to detect an encoder failure when the axis is not profiling.
- The IL command must be set to a value greater than the OV setting
- The TL command must be set to a value greater than the OV setting

Example:

The A axis is setup with the following settings for encoder failure detection:

```
OA 1
OT 500
OV 3
OE 1
ER 1000
```

The A axis is commanded to move 300 counts, but the B channel on the encoder has failed and no longer operates. Because the ER setting is greater than the commanded move, the error will not be detected by using the OE and ER commands, but this condition will be detected as a encoder failure. When the axis is commanded to move a 300 counts, the position error will cause the motor command voltage to be increased to a value that will be greater than the OV value, 3 volts in this case. Once the motor command output is greater than the OV threshold for more than than the 500ms defined by the OT command AND there has been less than 4 counts of change on the encoder, then the controller will turn off that axis due to an encoder failure. The motor will have moved some distance during this operation, but it will be shut down before a full runaway condition occurs.

Using Encoder Failure to detect a hard stop or stalled motor

The encoder failure detection can also be used to detect when an axis is up against a hard stop. In this scenario the motor command will be commanded above the OV threshold, but because the motor is not moving the controller will detect this scenario as an encoder failure.

Programmable Position Limits

The DMC-42x0 provides programmable forward and reverse position limits. These are set by the BL and FL software commands. Once a position limit is specified, the DMC-42x0 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example:

```
DP0,0,0                 Define Position
BL -2000,-4000,-8000    Set Reverse position limit
FL 2000,4000,8000       Set Forward position limit
JG 2000,2000,2000       Jog
BG XYZ                  Begin
```

(motion stops at forward limits)

Off-On-Error

The DMC-42x0 controller has a built in function which can turn off the motors under certain error conditions. This function is known as 'Off-On-Error". To activate the OE function for each axis, specify 1, 2 or 3 for that axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. A hardware limit is reached
3. The abort command is given
4. The abort input is activated with a low signal.

Note: If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

Examples:

```
OE 1,1,1,1      Enable off-on-error for X,Y,Z and W
OE 0,1,0,1      Enable off-on-error for Y and W axes and disable off-on-error for W and Z axes
OE 2,3          Enable off-on-error for limit switch for the X axis, and position error (or abort input) and limit switch for the Y axis
```

Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER, a encoder failure is detected, or the abort input is triggered. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

NOTE: The Error Subroutine will be entered again unless the error condition is cleared.

Example:

```
#A;JP #A;EN          "Dummy" program
#POSERR              Start error routine on error
MG "error"           Send message
SB 1                 Fire relay
STX                  Stop motor
AMX                  After motor stops
SHX                  Servo motor here to clear error
RE                   Return to main program
```

Limit Switch Routine

The DMC-42x0 provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. X,Y,Z, or W following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Limit Switch Example:

```
#A;JP #A;EN              Dummy Program
#LIMSWI                  Limit Switch Utility
V1=_LFX                  Check if forward limit
V2=_LRX                  Check if reverse limit
JP#LF,V1=0               Jump to #LF if forward
JP#LR,V2=0               Jump to #LR if reverse
JP#END                   Jump to end
#LF                      #LF
MG "FORWARD LIMIT"       Send message
STX;AMX                  Stop motion
PR-1000;BGX;AMX          Move in reverse
JP#END                   End
#LR                      #LR
MG "REVERSE LIMIT"       Send message
STX;AMX                  Stop motion
PR1000;BGX;AMX           Move forward
#END                     End
RE                       Return to main program
```

Chapter 9 Troubleshooting

Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Stability and Compensation
3. Operation
4. Error Light (Red LED)

The various symptoms along with the cause and the remedy are described in the following tables.

Installation

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Motor runs away with no connections from controller to amplifier input.	Adjusting offset causes the motor to change speed.	1. Amplifier has an internal offset. 2. Damaged amplifier.	1. Adjust amplifier offset. Amplifier offset may also be compensated by use of the offset configuration on the controller (see the OF command). 2. Replace amplifier.
Motor is enabled even when MO command is given	The SH command disables the motor	1. The amplifier requires the a different Amplifier Enable setting on the Interconnect Module	Refer to Chapter 3 or contact Galil.
Unable to read main or auxiliary encoder input.	The encoder does not work when swapped with another encoder input.	1. Wrong encoder connections. 2. Encoder is damaged 3. Encoder configuration incorrect.	1. Check encoder wiring. For single ended encoders (CHA and CHB only) do not make any connections to the CHA- and CHB- inputs. 2. Replace encoder 3. Check CE command
Unable to read main or auxiliary encoder input.	The encoder works correctly when swapped with another encoder input.	1. Wrong encoder connections. 2. Encoder configuration incorrect. 3. Encoder input or controller is damaged	1. Check encoder wiring. For single ended encoders (MA+ and MB+ only) do not make any connections to the MA- and MB- inputs. 2. Check CE command 3. Contact Galil
Encoder Position Drifts	Swapping cables fixes the problem	1. Poor Connections / intermittent cable	Review all connections and connector contacts.

Encoder Position Drifts	Significant noise can be seen on MA+ and / or MB+ encoder signals	1. Noise	Shield encoder cables Avoid placing power cables near encoder cables Avoid Ground Loops Use differential encoders Use ±12V encoders

Stability

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Servo motor runs away when the loop is closed.	Reversed Motor Type corrects situation (MT -1)	1. Wrong feedback polarity.	Reverse Motor or Encoder Wiring (remember to set Motor Type back to default value: MT 1)
Motor oscillates.		2. Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Controller rejects commands.	Response of controller from TC1 diagnoses error.	1. Anything	Correct problem reported by TC1
Motor Doesn't Move	Response of controller from TC1 diagnoses error.	2. Anything	Correct problem reported by SC

Error Light (Red LED)

The red error LED has multiple meanings for Galil controllers. Here is a list of reasons the error light will come on and possible solutions:

Under Voltage

If the controller is not receiving enough voltage to power up.

Under Current

If the power supply does not have enough current, the red LED will cycle on and off along with the green power LED.

Position Error

If any axis that is set up as a servo (MT command) has a position error value (TE) that exceeds the error limit (ER) - the error light will come on to signify there is an axis that has exceeded the position error limit. Use a DP*=0 to set all encoder positions to zero or a SH (Servo Here) command to eliminate position error.

Invalid Firmware

If the controller is interrupted during a firmware update or an incorrect version of firmware is installed - the error light will come on. The prompt will show up as a greater than sign ">" instead of the standard colon ":" prompt. Use GalilTools software to install the correct version of firmware to fix this problem.

Self Test

During the first few seconds of power up, it is normal for the red LED to turn on while it is performing a self test. If the self test detects a problem such as corrupted memory or damaged hardware - the error light will stay on to signal a problem with the board. To fix this problem, a Master Reset may be required. The Master Reset will set

the controller back to factory default conditions so it is recommended that all motor and I/O cables be removed for safety while performing the Master Reset. Cables can be plugged back in after the correct settings have been loaded back to the controller (when necessary). To perform a Master Reset - find the jumper location labeled MR or MRST on the controller and put a jumper across the two pins. Power up with the jumper installed. The Self-Test will take slightly longer - up to 5seconds. After the error light shuts off, it is safe to power down and remove the Master Reset jumper. If performing a Master Reset does not get rid of the error light, the controller may need to be sent back to the factory to be repaired. Contact Galil for more information.

Chapter 10 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Figure 10.1.

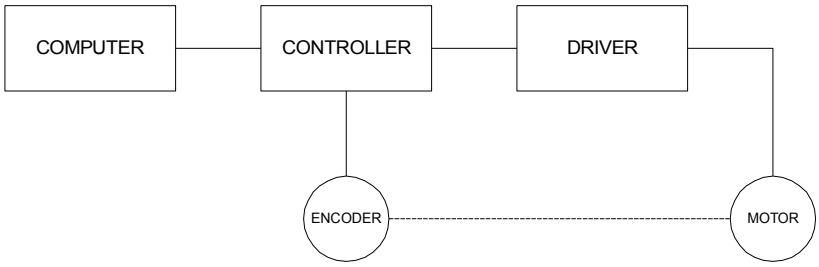


Figure 10.1: Elements of Servo Systems

The operation of such a system can be divided into three levels, as illustrated in Figure 10.2. The levels are:

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

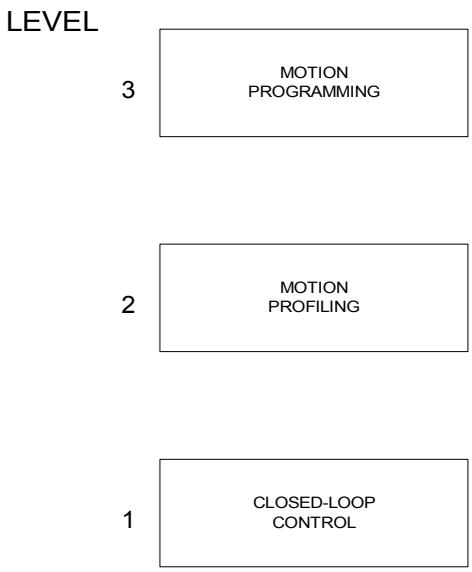


Figure 10.2: Levels of Control Funtions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000,4000
SP 20000,20000
AC 200000,00000
BG X
AD 2000
BG Y
EN
```

This program corresponds to the velocity profiles shown in Figure 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

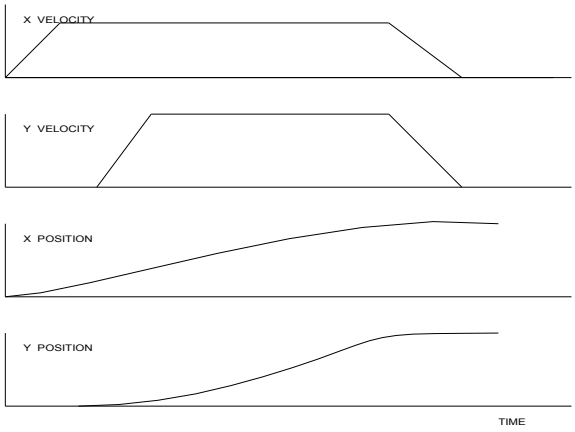


Figure 10.3: Velocity and Position Profiles

Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the “right” rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called over-damped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter which is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Figure 10.4. The mathematical model of the various components is given below.

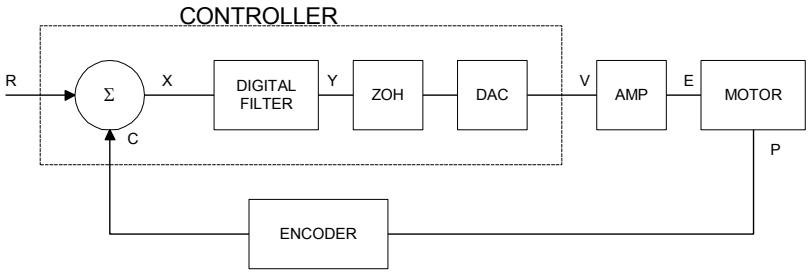


Figure 10.4: Functional Elements of a Motion Control System

Motor-Amplifier

The motor amplifier may be configured in three modes:

1. Voltage Drive
2. Current Drive
3. Velocity Loop

The operation and modeling in the three modes is as follows:

Voltage Drive

The amplifier is a voltage source with a gain of K_v [V/V]. The transfer function relating the input voltage, V, to the motor position, P, is

$$P/V = K_V / \left[K_t S (ST_m + 1)(ST_e + 1) \right]$$

where

$$T_m = RJ/K_t^2 \quad [s]$$

and

$$T_e = L/R \qquad [s]$$

and the motor parameters and units are

K_t	Torque constant [Nm/A]
R	Armature Resistance Ω
J	Combined inertia of motor and load [kg.m^2]
L	Armature Inductance [H]

When the motor parameters are given in English units, it is necessary to convert the quantities to MKS units. For example, consider a motor with the parameters:

K_t = 14.16 oz - in/A = 0.1 Nm/A

R = 2 Ω

$J = 0.0283$ oz-in-$s^2 = 2 * 10^{-4}$ kg . m^2

$L = 0.004H$

Then the corresponding time constants are

$T_m = 0.04$ sec

and

$T_e = 0.002$ sec

Assuming that the amplifier gain is $K_v = 4$, the resulting transfer function is

$P/V = 40/[s(0.04s+1)(0.002s+1)]$

Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of K_a. The resulting transfer function in this case is

$P/V = K_a K_t / Js^2$

where Kt and J are as defined previously. For example, a current amplifier with $K_a = 2$ A/V with the motor described by the previous example will have the transfer function:

$P/V = 1000/s^2$ [rad/V]

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is illustrated in Figure 10.5. Note that the transfer function between the input voltage V and the velocity ω is:

$\omega /V = [K_a K_t/Js]/[1+K_a K_t K_g/Js] = 1/[K_g(sT_1+1)]$

where the velocity time constant, T_1, equals

$T_1 = J/K_a K_t K_g$

This leads to the transfer function

$P/V = 1/[K_g s(sT_1+1)]$

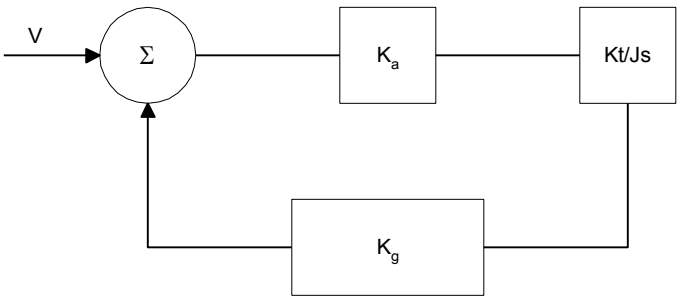


Figure 10.5: Elements of velocity loops

The resulting functions derived above are illustrated by the block diagram of Figure 10.6.

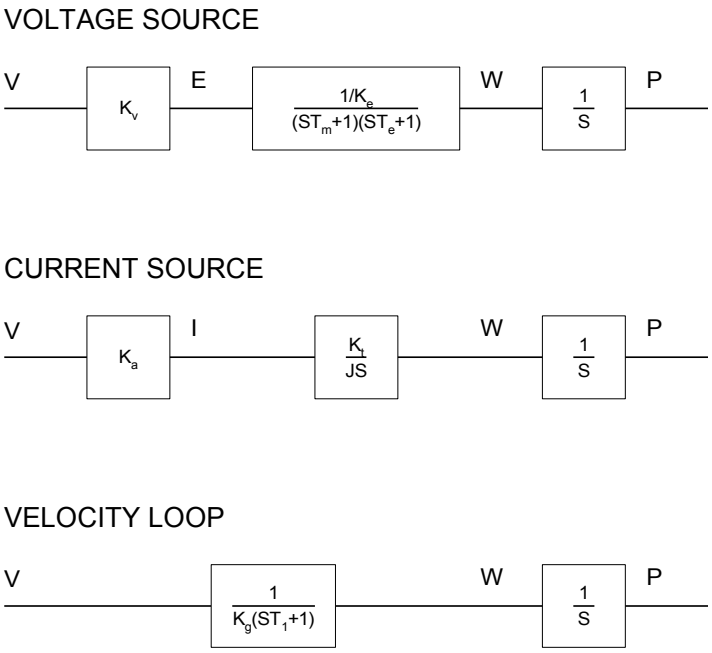


Figure 10.6: Mathematical model of the motor and amplifier in three operational modes

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$K_f = 4N/2\pi$ [count/rad]

For example, a 1000 lines/rev encoder is modeled as

$K_f = 638$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is ±10V or 20V. Therefore, the effective gain of the DAC is

K= 20/65536 = 0.0003 [V/count]

Digital Filter

The digital filter has three element in series: PID, low-pass and a notch filter. The transfer function of the filter. The transfer function of the filter elements are:

PID $D(z) = \frac{K(Z-A)}{Z} + \frac{CZ}{Z-1}$

Low-pass $L(z) = \frac{1-B}{Z-B}$

Notch $N(z) = \frac{(Z-z)(Z-\bar{z})}{(Z-p)(Z-\bar{p})}$

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD)
A = KD/(KP + KD)
C = KI
B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

$G(s) = (P + sD + I/s) \cdot a / (s + a)$

where,

P = KP
D = T·KD
I = KI/T

$a = \frac{1}{T} ln\left(\frac{1}{B}\right)$

where T is the sampling period, and B is the pole setting

For example, if the filter parameters of the DMC-42x0 are

KP = 16
KD = 144
KI = 2
PL = 0.75
T = 0.001 s

the digital filter coefficients are

K = 160
A = 0.9
C = 2
a = 250 rad/s

and the equivalent continuous filter, G(s), is

$G(s) = [16 + 0.144s + 2000/s] \cdot 250/ (s+250)$

The notch filter has two complex zeros, z and , and two complex poles, p and .

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient damping. It is best to select the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The most simple procedure for setting the notch filter, identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$H(s) = 1/(1+sT/2)$

If the sampling period is T = 0.001, for example, H(s) becomes:

$H(s) = 2000/(s+2000)$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the DMC-42x0 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2 * 10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	Ω	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer function of the system elements are:

Motor

$M(s) = P/I = K_t/Js^2 = 500/s^2$ [rad/A]

Amp

$K_a = 4$ [Amp/V]

DAC

$K_d = 0.0003$ [V/count]

Encoder

$K_f = 4N/2\pi = 318$ [count/rad]

ZOH

2000/(s+2000)

Digital Filter

KP = 12.5, KD = 245, T = 0.001

Therefore,

$D(z) = 1030 (z-0.95)/Z$

Accordingly, the coefficients of the continuous filter are:

P = 50

D = 0.98

The filter equation may be written in the continuous equivalent form:

$$G(s) = 50 + 0.98s = .098\ (s+51)$$

The system elements are shown in Figure 10.7.

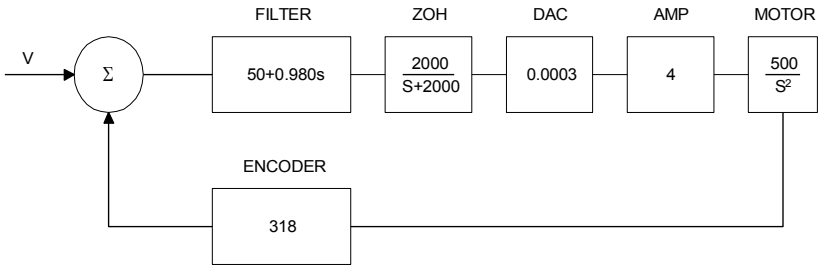


Figure 10.7: Mathematical model of the control system

The open loop transfer function, A(s), is the product of all the elements in the loop.

$$A(s) = 390{,}000\ (s+51)/[s^2(s+2000)]$$

To analyze the system stability, determine the crossover frequency, ω_c at which $A(j\ \omega_c)$ equals one. This can be done by the Bode plot of $A(j\ \omega_c)$, as shown in Figure 10.8.

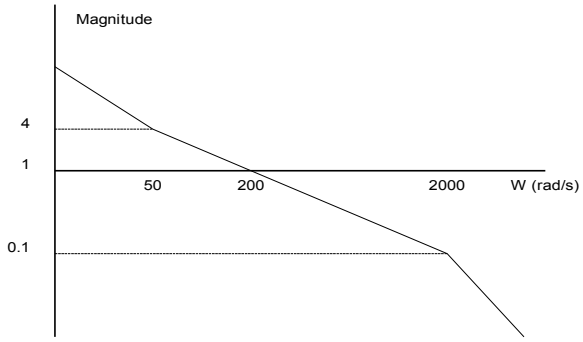


Figure 10.8: Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2\ .\ (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30° and 45°. The phase margin of 70° given above indicated over-damped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is pre-programmed in the DMC-42x0 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

K_t	Nm/A	Torque constant
$J = 2 * 10^{-4}$	$kg.m^2$	System moment of inertia
$R = 2$	Ω	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of the DMC-42x0 outputs ±10V for a 16-bit command of ±32768 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of $\omega_C = 500$ rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor

$M(s) = P/I = Kt/Js^2 = 1000/s^2$

Amp

$K_a = 2$ [Amp/V]

DAC

$K_d = 10/32768 = .0003$

Encoder

$K_f = 4N/2\pi = 636$

ZOH

$H(s) = 2000/(s+2000)$

Compensation Filter

$G(s) = P + sD$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

$$L(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 3.17 \cdot 10^6/[s^2(s+2000)]$$

Then the open loop transfer function, A(s), is

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_C = 500$.

$$L(j500) = 3.17 \cdot 10^6/[(j500)^2\, (j500+2000)]$$

This function has a magnitude of

$$|L(j500)| = 0.00625$$

and a phase

$$\text{Arg}[L(j500)] = -180° - \tan^{-1}(500/2000) = -194°$$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45°. This requires that

$$|A(j500)| = 1$$
$$\text{Arg}\,[A(j500)] = -135°$$

However, since

$$A(s) = L(s)\,G(s)$$

then it follows that G(s) must have magnitude of

$$|G(j500)| = |A(j500)/L(j500)| = 160$$

and a phase

$$\arg\,[G(j500)] = \arg\,[A(j500)] - \arg\,[L(j500)] = -135° + 194° = 59°$$

In other words, we need to select a filter function G(s) of the form

$$G(s) = P + sD$$

so that at the frequency ω_c =500, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

$$\arg\,[G(j500)] = \tan^{-1}[500D/P] = 59°$$

The solution of these equations leads to:

$$P = 160 \cos 59° = 82.4$$
$$500D = 160 \sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.2744s$$

The function G is equivalent to a digital filter of the form:

$$D(z) = KP + KD(1-z^{-1})$$

where

$$P = KP$$

$$D = KD \cdot T$$

and

$$KD = D/T$$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

$$KP = 82.4$$

$$KD = 247.4$$

The DMC-42x0 can be programmed with the instruction:

```
KP 82.4
KD 68.6
```

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

Equivalent Filter Form - DMC-42x0

Digital	$D(z) = [K(z-A/z) + Cz/(z-1)]\cdot(1-B)/(Z-B)$
KP, KD, KI, PL	$K = (KP + KD)$
	$A = KD/(KP+KD)$
	$C = KI$
	$B = PL$
Digital	$D(z) = [KP + KD(1-z^{-1}) + KI/2(1-z^{-1})] \cdot(1-PL)/(Z-PL)$
Continuous	$G(s) = (P + Ds + I/s)\cdot a/(s+a)$
PID, T	$P = KP$
	$D = T * KD$
	$I = KI / T$
	$a = 1/T \ln(1/PL)$

Appendices

Electrical Specifications

NOTE	Electrical specifications are only valid once controller is out of reset.

Servo Control

Motor command line	±10 V analog signal Resolution: 16-bit DAC or 0.0003 volts 3 mA maximum. Output impedance – 500 Ω
Main and auxiliary encoder inputs	TTL compatible, but can accept up to ±12 volts Quadrature phase on CHA, CHB Single-ended or differential Maximum A, B edge rate: 22 MHz Minimum IDX pulse width: 45 nsec

Stepper Control

STPn (Step)	TTL (0-5 volts) level at 50% duty cycle. 6,000,000 pulses/sec maximum frequency
DIRn (Direction)	TTL (0-5 volts)

Input / Output

Opto-isolated Inputs: DI[16:1]*, Limit switches, home, abort, reset	2.2 kΩ in series with opto-isolator Active high or low requires at least 1mA to activate. Once activated, the input requires the current to go below 0.5mA. All Limit Switch and Home inputs use one common voltage (LSCOM) which can accept up to 24 volts. Voltages above 24 volts require an additional resistor. ≥ 1 mA = ON; ≤ 0.5 mA = OFF *[8:1] for 1-4 axes models, [16:1] for 5-8 axes models

Analog Inputs: AI[8:1]	±10 volts 12-Bit Analog-to-Digital converter 16-bit optional
Optoisolated Digital Outputs: DO[16:1]*	500mA Sourcing *[8:1] for 1-4 axes models, [16:1] for 5-8 axes models
Extended I/O: IO[80:17]	Configurable 0-5V TTL as Inputs or Outputs Configured by the CO command in banks of 8
Auxiliary Inputs as Uncommitted Inputs: DI[96:81]*	The auxiliary pins can be used as uncommitted inputs and are assigned to the following bits: Axis A: DI81, DI82 Axis B: DI83, DI84 Axis C: DI85, DI86 Axis D: DI87, DI88 Axis E: DI89, DI90 Axis F: DI91, DI92 Axis G: DI93, DI94 Axis H: DI95, DI96 These inputs have the same specifications as listed above for encoder inputs. *The number of auxiliary inputs is dependent on the number of axes ordered

Power Requirements

20-80 V_{DC}	12-16 W at 25° C

+5, ±12V Power Output Specifications

Output Voltage	Tolerance	Max Current Output
+5V	±5%	1.1A
+12V	±5%	40mA
-12V	±5%	40mA

Performance Specifications

Minimum Servo Loop Update Time/Memory:

	Normal
Minimum Servo Loop Update Time	
DMC-4210	62.5 µsec
DMC-4220	62.5 µsec
DMC-4230	125 µsec
DMC-4240	125 µsec
DMC-4250	156.25 µsec
DMC-4260	156.25 µsec
DMC-4270	187.5 µsec
DMC-4280	187.5 µsec
Position Accuracy	±1 quadrature count
Velocity Accuracy	
Long Term	Phase-locked, better than 0.005%
Short Term	System dependent
Position Range	±2147483647 counts per move
Velocity Range	Up to 22,000,000 counts/sec servo; 6,000,000 pulses/sec-stepper
Velocity Resolution	2 counts/sec
Motor Command Resolution	16 bit or 0.0003 V
Variable Range	±2 billion
Variable Resolution	1×10^{-4}
Number of Variables	510
Array Size	24000 elements, 30 arrays
Program Size	4000 lines x 80 characters
Command Processing	~40 µsec per command

Environmental

Operating Temperature	0-70 deg C
Humidity	20-95% RH, non-condensing

Ordering Options

Overview

The DMC-42x0 can be ordered in many different internal boards, the DMC, ICM, and CMB are required, and the AMP/SMD are optional. These individual boards have their own set of options that can modify them. This section provides information regarding the different options available for these different boards. For information on pricing and how to order a controller with these options, see our DMC-42x0 part number generator on our website.

http://www.galilmc.com/products/dmc-42x0-part-number.php

DMC, "DMC-42x0(Y)" Controller Board Options

The following options are the "Y" configuration options that can be added to the DMC-42x0 part number. Multiple Y-options can be ordered per board.

-16 bit – 16 bit Analog Inputs

The -16 bit option provides 16 bit analog inputs on the DMC-42x0 motion controller. The standard resolution of the analog inputs is 12 bits.

Part number ordering example: DMC-4210(-16bit)-C012-I000

4-20mA – 4-20mA analog inputs

The 4-20mA option converts all 8 analog inputs into 4-20mA analog inputs. This is accomplished by installing 475W precision resistors between the analog inputs and ground. When using this option the analog inputs should be configured for 0-10V analog inputs using the AQ command (AQ n,4). The equation for calculating the current is:

$$I_{ma} = 2.105\ V$$

Where I_{ma} = current in mA

V = Voltage reading from DMC-42x0

Part number ordering example: DMC-4210(4-20mA)-C012-I000

ETL – ETL Certified DMC-42x0

The DMC-42x0 can be ordered in a configuration that is ETL listed. ETL Mark is shown in Figure A.1.

Figure A.1: ETL Mark with (ETL) Option

Part number ordering example: DMC-4210(ETL)-C012-I000

RS-422-Main Port

Standard connector and cable when DMC-42x0 is ordered with RS-422 Option.

Pin #	Signal
1	RTS-
2	TXD-
3	RXD-
4	CTS-
5	GND
6	RTS+
7	TXD+
8	RXD+
9	CTS+

RS-422-Auxiliary Port

Standard connector and cable when DMC-42x0 is ordered with RS-422 Option.

Pin #	Signal
1	CTS-
2	RXD-
3	TXD-
4	RTS-
5	GND
6	CTS+
7	RXD+
8	TXD+
9	RTS+

RS-232/422 Configuration Jumpers

Location	Label	Function (If jumpered)
JP2 (-C012, default) JP3 (-C022)	ARXD	Connects a 120 Ω Termination resistor between the differential “Receive” inputs on the Aux Serial port. Pins 2 and 7 on RS-422 Auxiliary Port.
	ACTS	Connects a 120 Ω Termination resistor between the differential “Clear To Send” inputs on the Aux Serial port. Pins 1 and 6 on RS-422 Auxiliary Port.
JP2 (-C012, default) JP2 (-C022)	MRXD	Connects a 120 Ω Termination resistor between the differential “Receive” inputs on the Main Serial port. Pins 3 and 8 on RS-422 Main Port.
	MCTS	Connects a 120 Ω Termination resistor between the differential “Clear To Send” inputs on the Main Serial port. Pins 4 and 9 on RS-422 Main Port.
JP2 (-CO12, default) JP3 (-C022)	APWR	Do **not** use with RS-422 option.

Note: The ARXD, ACTS, MRXD and MCTS should be installed for single-drop RS-422. For multi-drop, the jumpers should be installed on the last device.

SSI and BiSS – SSI and BiSS Absolute encoder Option

The BiSS and SSI options configures the DMC-42x0 for BiSS or SSI absolute encoder inputs. See the SS and SI commands in the DMC-42x0 Command Reference. Pin-out information is shown below for the DMC-42x0 with the CB-50-80, and the regular 80 Pin High Density cable.

DMC-42x0 (BiSS or SSI Option) Pinout

SSI Signal	DMC-42x0 80-Pin SCSI	DMC-42x0 with CB-50-80
D0-	44	J6-4
-D0+	42	J6-2
D1-	48	J6-8
D1+	46	J6-6
D2-	52	J6-12
D2+	50	J6-10
D3-	56	J6-16
D3+	54	J6-14
D4-	4	J8-4
D4+	2	J8-2
D5-	8	J8-8
D5+	6	J8-6
D6-	12	J8-12
D6+	10	J8-10
D7-	16	J8-16
D7+	14	J8-14
C0-	71	J6-31
C0+	69	J6-29
C1-	67	J6-29
C1+	65	J6-25
C2-	63	J6-23
C2+	61	J6-21
C3-	59	J6-19
C3+	57	J6-17
C4-	31	J8-31
C4+	29	J8-29
C5-	27	J8-27
C5+	25	J8-25
C6-	23	J8-23
C6+	21	J8-21
C7-	19	J8-19
C7+	17	J8-17
VCC (+5V)	40,80	J6-49, J8-49
GND	18,20,22,24,26,28,30,58,60,62,64,66,68,7	J6-10>50 (EVEN), J8-10>50 (EVEN)

Note: This option is not valid with the ICM-42100 (-I100). Consult Galil if the Sinusoidal encoder and SSI encoder interfaces are required on the same set of 4 axes.

Part number ordering example: DMC-4210(SSI)

TRES – Encoder Termination Resistors

The TRES option provides termination resistors on all of the main and auxiliary encoder inputs on the DMC-42x0 motion controller. The termination resistors are 120 Ω, and are placed between the positive and negative differential inputs on the Main A, B, Index channels as well as the Auxiliary A and B channels as in Figure A.2.

Note: Single ended encoders will not operate correctly with the termination resistors installed. If a combination of differential encoder inputs with termination resistors and single ended encoders is required on the same interconnect module, contact Galil directly.

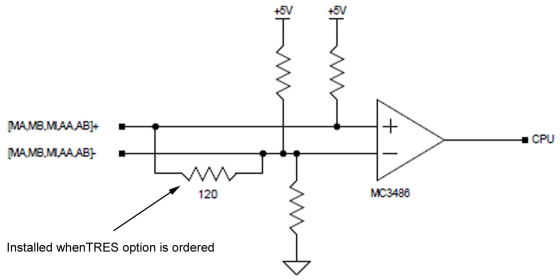


Figure A.2: Encoder Inputs with -TRES option

Part number ordering example: DMC-4210(TRES)

Accessories

CABLE-100-1M	100-pin high density cable, 1 meter
CABLE-100-4M	100-pin high density cable, 4 meter
CABLE-80-1M	80-pin high density cable, 1 meter
CABLE-80-4M	80-pin high density cable, 4 meter
CABLE-36-1M	36-pin high density cable, 1 meter
CABLE-36-4M	36-pin high density cable, 4 meter
CB-50-100	50-pin to 100-pin converter board, includes two 50-pin ribbon cables
CB-50-80	50-pin to 80-pin converter board, includes two 50-pin ribbon cables
ICM-1900	Interconnect module
-LAEN	Option for ICM-1900, Provides Active Low Amplifier Enable Signal
-OPTO	Option for ICM-1900, Provides Opto0isolation for digital outputs
-OPTOHC	Option for ICM-1900, Provides High Current Opto-isolation for digital outputs
AMP-19x0	Interconnect module with 1 - 4 brushless/brush PWM motor amplifiers where x specifies the either 2 or 4 amplifiers.
-OPTO	Option for AMP-19x0, Provides Opto0isolation for digital outputs
-OPTOHC	Option for AMP-19x0, Provides High Current Opto-isolation for digital outputs
ICM-2900	Interconnect module with detachable screw terminal
-LAEN	Option for ICM-2900, Provides Active Low Amplifier Enable Signal
-FL	Option for ICM-2900 where the ICM-2900 includes flanges for rack mounting
-OPTO	Option for AMP-19x0, provides Opto-isolation for digital outputs
-OPTOHC	Option for AMP-19x0, provides High Current Opto-isolation for digital outputs

Input Current Limitations

The current for an optoisolated input shall not exceed 11mA. Some applications may require the use of an external resistor (R) to limit the amount of current for an input. These external resistors can be placed in series between the inputs and their power supply (Vs). To determine if an additional resistor (R) is required, follow Equation A1 below for guidance.

$$1\,mA < \frac{Vs}{R + 2200\,\Omega} < 11\,mA$$

Equation A1: Current limitation requirements for each input.

Serial Cable Connections

The DMC-42x0 requires the transmit, receive, and ground for slow communication rates. (i.e. 9600 baud) For faster rates the handshake lines are required. The connection tables below contain the handshake lines.

Standard RS-232 Specifications

25 pin Serial Connector (Male, D-type)

This table describes the pinout for standard serial ports found on most computers.

Pin #	Function
1	NC
2	TXD
3	RXD
4	RTS
5	CTS
6	DSR
7	GND
8	DCD
9	NC
10	NC
11	NC
12	NC
13	NC
14	NC
15	NC
16	NC
17	NC
18	NC
19	NC
20	DTR
21	NC
22	RI
23	NC
24	NC
25	NC

9 Pin Serial Connector (Male, D-type)

Standard serial port connections found on most computers.

Pin #	Function
1	DCD
2	RXD
3	TXD
4	RTS
5	GND
6	DSR
7	RTS
8	CTS
9	RI

DMC-42x0 Serial Cable Specifications

Cable to Connect Computer 25 pin to Main Serial Port

25 Pin (Male - computer)	9 Pin (female - controller)
5 CTS	8 RTS
3 RXD	2 TXD
2 TXD	3 RXD
4 RTS	7 CTS
7 GND	5 GND

Cable to Connect Computer 9 pin to Main Serial Port Cable (9 pin)

9 Pin (FEMALE - Computer)	9 Pin (FEMALE - Controller)
2 RXD	2 TXD
3 TXD	3 RXD
5 GND	5 GND
7 RTS	7 CTS
8 CTS	8 RTS

Cable to Connect Computer 25 pin to Auxiliary Serial Port Cable (9 pin)

25 Pin (Male - terminal)	9 Pin (male - controller)
4 RTS	8 CTS
2 TXD	2 RXD
3 RXD	3 TXD
5 CTS	7 RTS
7 GND	5 GND
Computer +5V	9 (Jumper APWR if required)

Cable to Connect Computer 9 pin to Auxiliary Serial Port Cable (9 pin)

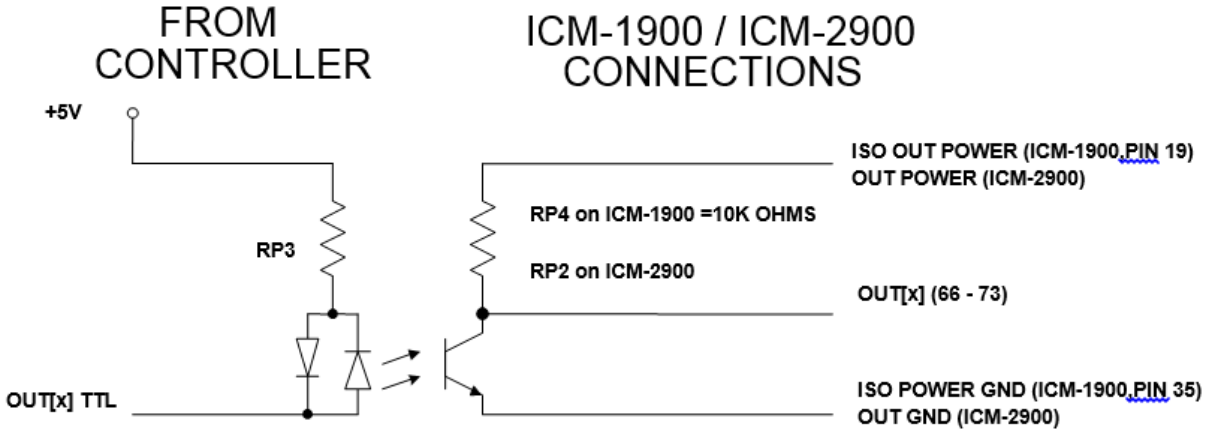
9 Pin (FEMALE - terminal)	9 Pin (MALE - Controller)
4 RTS	8 CTS
3 TXD	2 RXD
2 RXD	3 TXD
1 CTS	7 RTS
5 GND	5 GND
Controller +5V	9 (Jumper APWR if required)

Opto-Isolated Outputs for ICM-2900/ICM-1900/AMP-19520/40

The ICM/AMP 1900 and ICM-2900 modules from Galil have an option for opto-isolated outputs.

Standard Opto-Isolation and High Current Opto-isolation:

The Opto-isolation option on the ICM-1900 has 2 forms: -opto (standard) and -optohc (high current). The standard version provides outputs with 4ma drive current / output with approximately 2 usec response time. The high current version provides 25ma drive current / output with approximately 400 usec response time.



The ISO OUT POWER (OUT POWER ON ICM-2900) and ISO POWER GND (OUT GND ON ICM-2900) signals should be connected to an isolated power supply. This power supply should be used only to power the outputs in order to obtain isolation from the controller. The signal "OUT[x]" is one of the isolated digital outputs where X stands for the digital output terminals.

The default configuration is for active high outputs. If active low outputs are desired, reverse RP3 in it's socket. This will tie RP3 to GND instead of VCC, inverting the sense of the outputs.

NOTE: If power is applied to the outputs with an isolated power supply but power is not applied to the controller, the outputs will float high (unable to sink current). This may present a problem when using active high logic and care should be taken. Using active low logic should avoid any problems associated with the outputs floating high.

Signal Descriptions

Outputs

Motor Command	± 10 Volt range signal for driving amplifier. In servo mode, motor command output is updated at the controller sample rate. In the motor off mode, this output is held at the OF command level.
Amplifier Enable	Signal to disable and enable an amplifier. Amp Enable goes low on Abort and OE1.
PWM / Step	PWM/STEP OUT is used for directly driving power bridges for DC servo motors or for driving step motor amplifiers. For servo motors: If you are using a conventional amplifier that accepts a ±10 Volt analog signal, this pin is not used and should be left open. The PWM output is available in two formats: Inverter and Sign Magnitude. In the Inverter mode, the PWM (64kHz) signal is .2% duty cycle for full negative voltage, 50% for 0 Voltage and 99.8% for full positive voltage (64kHz Switching Frequency). In the Sign Magnitude Mode (MT1.5), the PWM (128 kHz) signal is 0% for 0 Voltage, 99.6% for full voltage and the sign of the Motor Command is available at the sign output (128kHz Switching Frequency).
PWM / Step	For stepper motors: The STEP OUT pin produces a series of pulses for input to a step motor driver. The pulses may either be low or high. The pulse width is 50%.
Sign / Direction	Used with PWM signal to give the sign of the motor command for servo amplifiers or direction for step motors.
Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 8 Output 9-Output 16 (DMC-4250 thru 4080)	The high power optically isolated outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

Inputs

Encoder, MA+, MB+	Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 22,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). Note: Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode.
Encoder Index, MI+	Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index.
Encoder, MA-, MB-, MI-	Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional.
Auxiliary Encoder, AA+, AB+, Aux A-, Aux B-	Inputs for additional encoder. Used when an encoder on both the motor and the load is required. Not available on axes configured for step motors.
Abort	A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program.
Reset	A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored.
Electronic Lock Out	Input that when triggered will shut down the amplifiers at a hardware level. Useful for safety applications where amplifiers must be shut down at a hardware level.

Forward Limit Switch	When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Reverse Limit Switch	When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Home Switch	Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command.
Input 1 - Input 8 isolated Input 9 - Input 16 isolated	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W if the high speed position latch function is enabled.
Latch	High speed position latch to capture axis position on occurrence of latch signal. AL command arms latch. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W. Input 9 is latch E, input 10 is latch F, input 11 is latch G, input 12 is latch H.

Connectors for DMC-42x0 Main Board

DMC-42x0 Axes A-D 100 pin High Density Connector

Pin	Signal	Pin	Signal
1	Analog Ground	51	NC
2	GND	52	GND
3	+5V	53	+5V
4	Error Output	54	Limit Common
5	Reset	55	Home W
6	Encoder-Compare Output	56	Reverse Limit W
7	GND	57	Forward Limit W
8	GND	58	Home Z
9	Motor Command W	59	Reverse Limit Z
10	Sign W/ Dir W	60	Forward Limit Z
11	PWM W / Step W	61	Home Y
12	Motor Command Z	62	Reverse Limit Y
13	Sign Z/ Dir Z	63	Forward Limit Y
14	PWM Z / Step Z	64	Home X
15	Motor Command Y	65	Reverse Limit X
16	Sign Y/ Dir Y	66	Forward Limit X
17	PWM Y / Step Y	67	GND
18	Motor Command X	68	+5V
19	Sign X/ Dir X	69	Input Common
20	PWM X/ Step X	70	Latch X / Digital Input 1

21	Amp Enable W	71	Latch Y / Digital Input 2
22	Amp Enable Z	72	Latch Z / Digital Input 3
23	Amp Enable Y	73	Latch W/ Digital Input 4
24	Amp Enable X	74	Digital Input 5
25	A+ X	75	Digital Input 6
26	A- X	76	Digital Input 7
27	B+ X	77	Digital Input 8
28	B- X	78	Abort
29	I+ X	79	Output 1
30	I- X	80	Output 2
31	A+ Y	81	Output 3
32	A- Y	82	Output 4
33	B+ Y	83	Output 5
34	B- Y	84	Output 6
35	I+ Y	85	Output 7
36	I- Y	86	Output 8
37	A+ Z	87	+5V
38	A- Z	88	GND
39	B+ Z	89	GND
40	B- Z	90	GND
41	I+ Z	91	Analog In 1
42	I- Z	92	Analog In 2
43	A+ W	93	Analog In 3
44	A- W	94	Analog In 4
45	B+ W	95	Analog In 5
46	B- W	96	Analog In 6
47	I+ W	97	Analog In 7
48	I- W	98	Analog In 8
49	+12V	99	-12V
50	+12V	100	-12V

DMC-42x0 Axes E-H 100 pin High Density Connector

Pin	Signal	Pin	Signal
1	NC	51	NC
2	GND	52	GND
3	+5V	53	+5V
4	Error Output	54	Limit Common
5	Reset	55	Home H

6	Encoder-Compare Output	56	Reverse Limit H
7	GND	57	Forward Limit H
8	GND	58	Home G
9	Motor Command H	59	Reverse Limit G
10	Sign H/ Dir H	60	Forward Limit G
11	PWM H / Step H	61	Home F
12	Motor Command G	62	Reverse Limit F
13	Sign G/ Dir G	63	Forward Limit F
14	PWM G / Step G	64	Home E
15	Motor Command F	65	Reverse Limit E
16	Sign F/ Dir F	66	Forward Limit E
17	PWM E / Step E	67	GND
18	Motor Command E	68	+5V
19	Sign E/ Dir E	69	Input Common
20	PWM E/ Step E	70	Latch E/ Digital Input 9
21	Amp Enable H	71	Latch F / Digital Input 10
22	Amp Enable G	72	Latch G / Digital Input 11
23	Amp Enable F	73	Latch H/ Digital Input 12
24	Amp Enable E	74	Digital Input 13
25	A+ E	75	Digital Input 14
26	A- E	76	Digital Input 15
27	B+ E	77	Digital Input 16
28	B- E	78	Abort
29	I+ E	79	Output 9
30	I- E	80	Output 10
31	A+ F	81	Output 11
32	A- F	82	Output 12
33	B+ F	83	Output 13
34	B- F	84	Output 14
35	I+ F	85	Output 15
36	I- F	86	Output 16
37	A+ G	87	+5V
38	A- G	88	GND
39	B+ G	89	GND
40	B- G	90	GND
41	I+ G	91	NC
42	I- G	92	NC
43	A+ H	93	NC
44	A- H	94	NC

45	B+ H	95	NC
46	B- H	96	NC
47	I+ H	97	NC
48	I- H	98	NC
49	+12V	99	-12V
50	+12V	100	-12V

DMC-42x0 Auxiliary Encoder 36 Pin High Density Connector

Pin	Signal	Pin	Signal
1	+5V	19	+5V
2	GND	20	GND
3	+AAX	21	+AAE
4	-AAX	22	-AAE
5	+ABX	23	+ABE
6	-ABX	24	-ABE
7	+AAY	25	+AAF
8	-AAY	26	-AAF
9	+ABY	27	+ABF
10	-ABY	28	-ABF
11	+AAZ	29	+AAG
12	-AAZ	30	-AAG
13	+ABZ	31	+ABG
14	-ABZ	32	-ABG
15	+AAW	33	+AAH
16	-AAW	34	-AAH
17	+ABW	35	+ABH
18	-ABW	36	-ABH

DMC-42x0 Extended I/O 80 Pin High Density Connector

Pin	Signal	Block	Bit @IN[n], @OUT[n]	Bit No.
1	I/O	8	72	7
2	I/O	9	73	0
3	I/O	8	71	6
4	I/O	9	74	1
5	I/O	8	70	5
6	I/O	9	75	2
7	I/O	8	69	4

8	I/O	9	76	3
9	I/O	8	68	3
10	I/O	9	77	4
11	I/O	8	67	2
12	I/O	9	78	5
13	I/O	8	66	1
14	I/O	9	79	6
15	I/O	8	65	0
16	I/O	9	80	7
17	I/O	7	64	7
18	GND	-	-	GND
19	I/O	7	63	6
20	GND	-	-	GND
21	I/O	7	62	5
22	GND	-	-	GND
23	I/O	7	61	4
24	GND	-	-	GND
25	I/O	7	60	3
26	GND	-	-	GND
27	I/O	7	59	2
28	GND	-	-	GND
29	I/O	7	58	1
30	GND	-	-	GND
31	I/O	7	57	0
32	I/O	6	56	7
33	I/O	6	55	6
34	I/O	6	54	5
35	I/O	6	53	4
36	I/O	6	52	3
37	I/O	6	51	2
38	I/O	6	50	1
39	I/O	6	49	0
40	+5V	-	-	+5V
41	I/O	4	40	7
42	I/O	5	41	0
43	I/O	4	39	6
44	I/O	5	42	1
45	I/O	4	38	5
46	I/O	5	43	2

47	I/O	4	37	4
48	I/O	5	44	3
49	I/O	4	36	3
50	I/O	5	45	4
51	I/O	4	35	2
52	I/O	5	46	5
53	I/O	4	34	1
54	I/O	5	47	6
55	I/O	4	33	0
56	I/O	5	48	7
57	I/O	3	32	7
58	GND	-	-	GND
59	I/O	3	31	6
60	GND	-	-	GND
61	I/O	3	30	5
62	GND	-	-	GND
63	I/O	3	29	4
64	GND	-	-	GND
65	I/O	3	28	3
66	GND	-	-	GND
67	I/O	3	27	2
68	GND	-	-	GND
69	I/O	3	26	1
70	GND	-	-	GND
71	I/O	3	25	0
72	I/O	2	24	7
73	I/O	2	23	6
74	I/O	2	22	5
75	I/O	2	21	4
76	I/O	2	20	3
77	I/O	2	19	2
78	I/O	2	18	1
79	I/O	2	17	0
80	+5V	-	-	+5V

RS-422-Main Port

Standard connector and cable when DMC-42x0 is ordered with RS-422 Option.

Pin #	Signal
1	RTS-

2	TXD-
3	RXD-
4	CTS-
5	GND
6	RTS+
7	TXD+
8	RXD+
9	CTS+

RS-422-Auxiliary Port

Standard connector and cable when DMC-42x0 is ordered with RS-422 Option.

Pin #	Signal
1	CTS-
2	RXD-
3	TXD-
4	RTS-
5	GND
6	CTS+
7	RXD+
8	TXD+
9	RTS+

Ethernet

The Ethernet connection is Auto MDIX, 100bT/10bT.

Pin	Signal
1	TXP
2	TXN
3	RXP
4	NC
5	NC
6	RXN
7	NC
8	NC

On the each Ethernet port there are two LEDs that indicate the status of the port's Ethernet connection.

Green Link LED (LNK): The green LED indicates there is a valid Ethernet connection. This LED will show that the physical Ethernet layer (the cable) is connected. This LED will also blink to show both transmit and receive activity across the connection.

Orange LED (SPD) : The orange LED indicates the speed of the Ethernet connection. It will be illuminated for a 100bT connection, and will be off for a 10bT connection.

A1. ICM-2900 Interconnect Module

Description

The ICM-2900 interconnect module provides easy connections between the Optima series controllers and other system elements, such as amplifiers, encoders, and external switches. The ICM- 2900 accepts the 100-pin main cable and provides terminal blocks for connections. Each terminal is labeled for quick connection of system elements. The ICM-2900 provides access to the signals for up to 4 axes (Two required for 5 or more axes).

Mechanical Specifications

Description	Specification
Weight	2.3 lb
Length	12.25 in.
Width	2.61 in.
Height	2.37 in.

Environmental Specifications

Description	Specification
Storage Temperature	-25° to +70 ° C
Operating Temperature	0° to +70 ° C
Operating Altitude	10,000 feet

Equipment Maintenance

The DMC-42x0 does not require maintenance.

Pinout

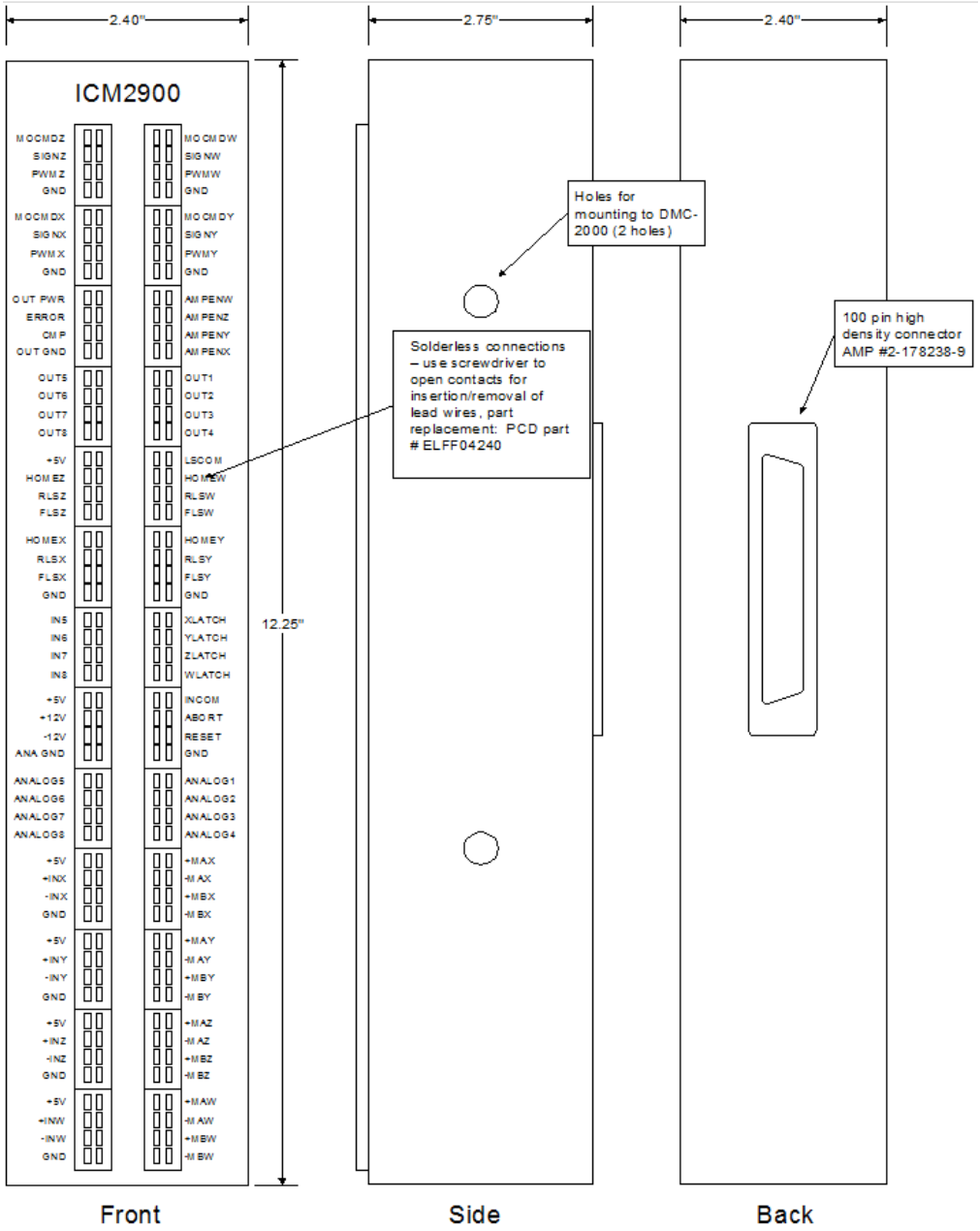
Block (4 PIN)	Label	I/O	Description
1	MOCMDZ	O	Z axis motor command to amp input (w / respect to ground)
1	SIGNZ	O	Z axis sign output for input to stepper motor amp
1	PWMZ	O	Z axis pulse output for input to stepper motor amp
1	GND	O	Signal Ground
2	MOCMDW	O	W axis motor command to amp input (w / respect to ground)
2	SIGNW	O	W axis sign output for input to stepper motor amp
2	PWMW	O	W axis pulse output for input to stepper motor amp
2	GND	O	Signal Ground
3	MOCMDX	O	X axis motor command to amp input (w / respect to ground)
3	SIGNX	O	X axis sign output for input to stepper motor amp
3	PWMX	O	X axis pulse output for input to stepper motor amp

3	GND	O	Signal Ground
4	MOCMDY	O	Y axis motor command to amp input (w / respect to ground)
4	SIGNY	O	Y axis sign output for input to stepper motor amp
4	PWMY	O	Y axis pulse output for input to stepper motor amp
4	GND	O	Signal Ground
5	OUT PWR	I	Isolated Power In for Opto-Isolation Option
5	ERROR	O	Error output
5	CMP	O	Circular Compare Output
5	OUT GND	O	Isolated Ground for Opto-Isolation Option
6	AMPENW	O	W axis amplifier enable
6	AMPENZ	O	Z axis amplifier enable
6	AMPENY	O	Y axis amplifier enable
6	AMPENX	O	X axis amplifier enable
7	OUT5	O	General Output 5
7	OUT6	O	General Output 6
7	OUT7	O	General Output 7
7	OUT8	O	General Output 8
8	OUT1	O	General Output 1
8	OUT2	O	General Output 2
8	OUT3	O	General Output 3
8	OUT4	O	General Output 4
9	+5V	O	+ 5 volts
9	HOMEZ	I	Z axis home input
9	RLSZ	I	Z axis reverse limit switch input
9	FLSZ	I	Z axis forward limit switch input
10	LSCOM	I	Limit Switch Common Input
10	HOMEW	I	W axis home input
10	RLSW	I	W axis reverse limit switch input
10	FLSW	I	W axis forward limit switch input
11	HOMEX	I	X axis home input
11	RLSX	I	X axis reverse limit switch input
11	FLSX	I	X axis forward limit switch input
11	GND	O	Signal Ground
12	HOMEY	I	Y axis home input
12	RLSY	I	Y axis reverse limit switch input
12	FLSY	I	Y axis forward limit switch input
12	GND	O	Signal Ground
13	IN5	I	Input 5
13	IN6	I	Input 6
13	IN7	I	Input 7
13	IN8	I	Input 8
14	XLATCH	I	Input 1 (Used for X axis latch input)
14	YLATCH	I	Input 2 (Used for Y axis latch input)

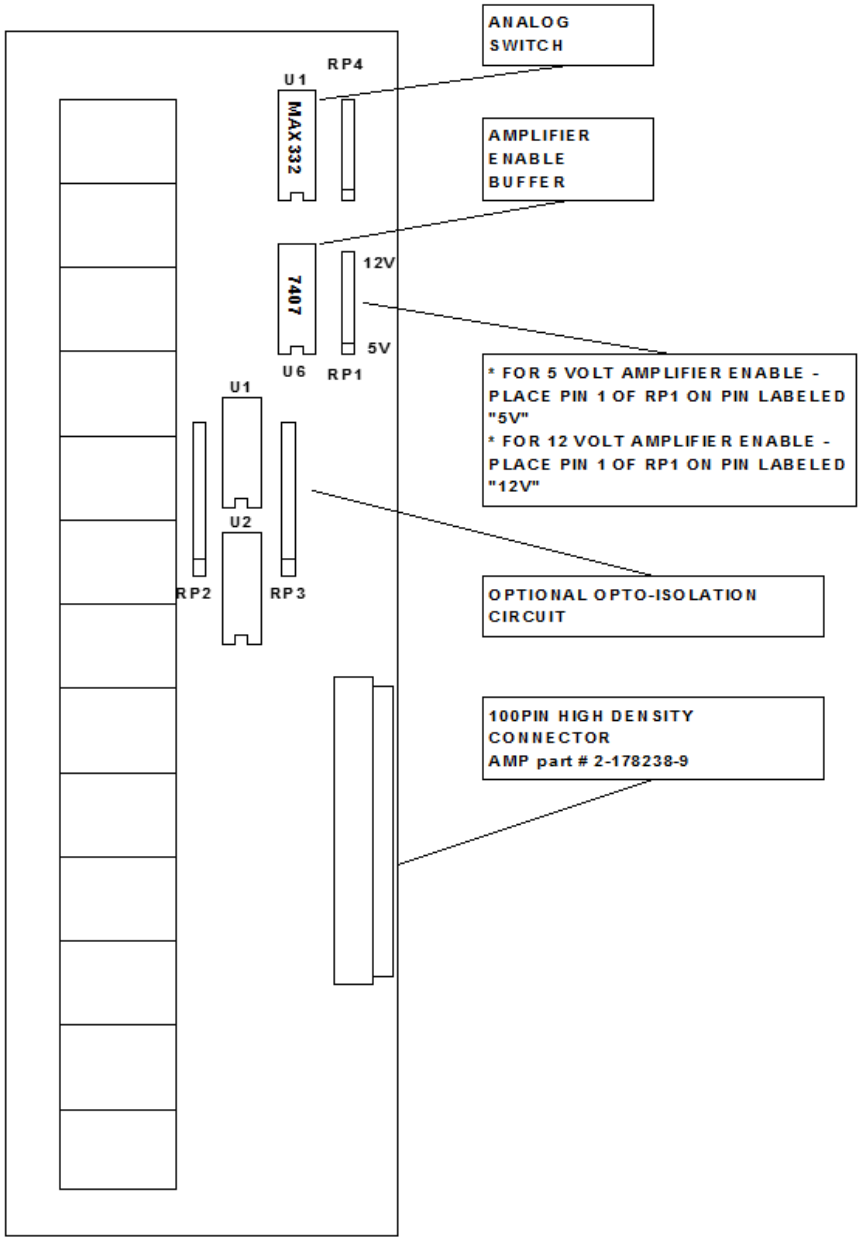
14	ZLATCH	I	Input 3 (Used for Z axis latch input)
14	WLATCH	I	Input 4 (Used for W axis latch input)
15	+5V	O	+ 5 volts
15	+12V	O	+12 volts
15	-12V	O	-12 volts
15	ANA GND	O	Isolated Analog Ground for Use with Analog Inputs
16	INCOM	I	Input Common For General Use Inputs
16	ABORT	I	Abort Input
16	RESET	I	Reset Input
16	GND	O	Signal Ground
17	ANALOG5	I	Analog Input 5
17	ANALOG6	I	Analog Input 6
17	ANALOG7	I	Analog Input 7
17	ANALOG8	I	Analog Input 8
18	ANALOG1	I	Analog Input 1
18	ANALOG2	I	Analog Input 2
18	ANALOG3	I	Analog Input 3
18	ANALOG4	I	Analog Input 4
19	+5V	O	+ 5 volts
19	+INX	I	X Main encoder Index +
19	-INX	I	X Main encoder Index -
19	GND	O	Signal Ground
20	+MAX	I	X Main encoder A+
20	-MAX	I	X Main encoder A-
20	+MBX	I	X Main encoder B+
20	-MBX	I	X Main encoder B-
21	+5V	O	+ 5 volts
21	+INY	I	Y Main encoder Index +
21	-INY	I	Y Main encoder Index -
21	GND	O	Signal Ground
22	+MAY	I	Y Main encoder A+
22	-MAY	I	Y Main encoder A-
22	+MBY	I	Y Main encoder B+
22	-MBY	I	Y Main encoder B-
23	+5V	O	+ 5 volts
23	+INZ	I	Z Main encoder Index +
23	-INZ	I	Z Main encoder Index -
23	GND	O	Signal Ground
24	+MAZ	I	Z Main encoder A+
24	-MAZ	I	Z Main encoder A-
24	+MBZ	I	Z Main encoder B+
24	-MBZ	I	Z Main encoder B-
25	+5V	O	+ 5 volts

25	+INW	I	W Main encoder Index +
25	-INW	I	W Main encoder Index -
25	GND	O	Signal Ground
26	+MAW	I	W Main encoder A+
26	-MAW	I	W Main encoder A-
26	+MBW	I	W Main encoder B+
26	-MBW	I	W Main encoder B

ICM-2900 Drawing



ICM-2900 PCB Layout



A2. ICM-2908 Interconnect Module

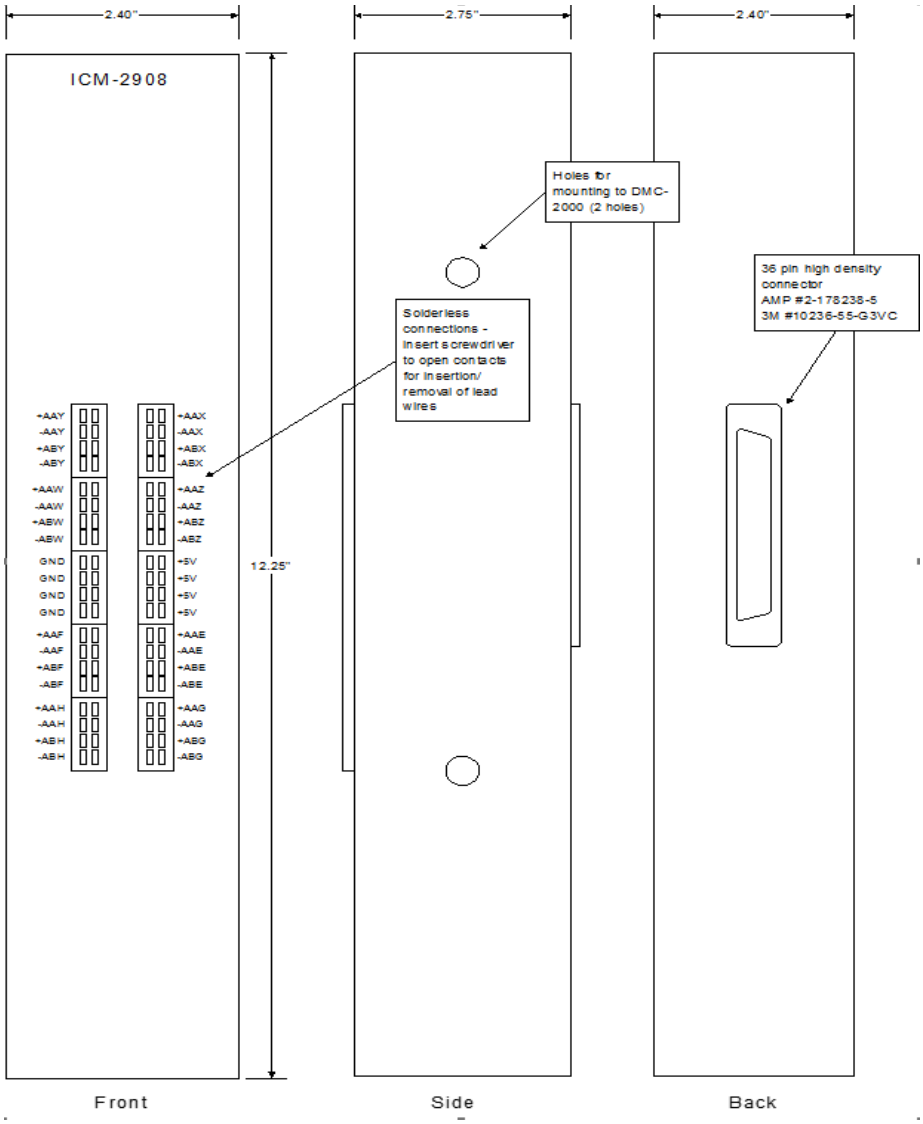
Description

The ICM-2908 interconnect module provides easy connections between the auxiliary encoder connections of the DMC-42x0 series controller and other system elements. The ICM-2908 accepts the 36 pin high density cable (CABLE-36) from the controller and provides terminal blocks for easy access. Each terminal is labeled for quick connection. One ICM-1908 provides access to all of the auxiliary encoders on a DMC-42x0 (up to 8 axes). ICM-2908 Drawing

For more information please refer to the following Application Note:

http://www.galil.com/download/application-note/note1428.pdf

ICM-2908 Drawing



A3. ICM-1900 Interconnect Module

Description

The ICM-1900 interconnect module provides easy connections between the DMC-42x0 series controllers and other system elements, such as amplifiers, encoders, and external switches. The ICM- 1900 accepts the 100-pin main cable and 25-pin auxiliary cable and breaks them into screw-type terminals. Each screw terminal is labeled for quick connection of system elements. An ICM-1900 is required for each set of 4 axes. (Two required for DMC-4250 thru DMC-4280).

The ICM-1900 is contained in a metal enclosure. A version of the ICM-1900 is also available with servo amplifiers (see AMP-19x0).

Features

- Separate DMC-42x0 cables into individual screw-type terminals
- Clearly identifies all terminals
- Provides jumper for connecting limit and input supplies to 5 V supply from PC
- Available with on-board servo drives (see AMP-19X0)
- Can be configured for AEN high or low

NOTE: The part number for the 100-pin connector is #2-178238-9 from AMP

Pinout

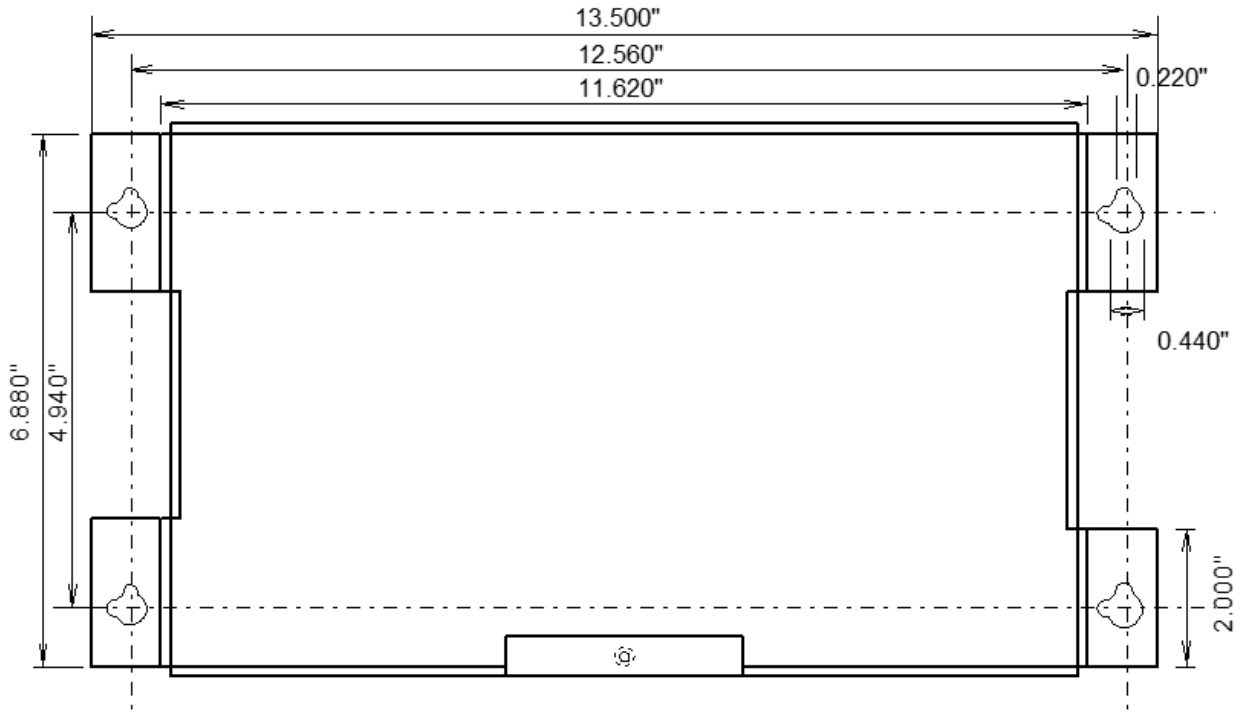
Terminal	Label	I/O	Description
1	+AAX	I	X Auxiliary encoder A+
2	-AAX	I	X Auxiliary encoder A-
3	+ABX	I	X Auxiliary encoder B+
4	-ABX	I	X Auxiliary encoder B-
5	+AAY	I	Y Auxiliary encoder A+
6	-AAY	I	Y Auxiliary encoder A-
7	+ABY	I	Y Auxiliary encoder B+
8	-ABY	I	Y Auxiliary encoder B-
9	+AAZ	I	Z Auxiliary encoder A+
10	-AAZ	I	Z Auxiliary encoder A-
11	+ABZ	I	Z Auxiliary encoder B+
12	-ABZ	I	Z Auxiliary encoder B-
13	+AAW	I	W Auxiliary encoder A+
14	-AAW	I	W Auxiliary encoder A-

15	+ABW	I	W Auxiliary encoder B+
16	-ABW	I	W Auxiliary encoder B-
17	GND		Signal Ground
18	+VCC		+ 5 volts
19	ISO OUT POWER	O	Isolated Output Power(for use with the opto-isolated output option)
20	ERROR	O	Error signal
21	RESET	I	Reset
22	CMP	O	Circular Compare output
23	MOCMDW	O	W axis motor command to amp input (w / respect to ground)
24	SIGNW	O	W axis sign output for input to stepper motor amp
25	PWMW	O	W axis pulse output for input to stepper motor amp
26	MOCMDZ	O	Z axis motor command to amp input (w / respect to ground)
27	SIGNZ	O	Z axis sign output for input to stepper motor amp
28	PWMZ	O	Z axis pulse output for input to stepper motor amp
29	MOCMDY	O	Y axis motor command to amp input (w / respect to ground)
30	SIGNY	O	Y axis sign output for input to stepper motor amp
31	PWMY	O	Y axis pulse output for input to stepper motor amp
32	MOCMDX	O	X axis motor command to amp input (w / respect to ground)
33	SIGNX	O	X axis sign output for input to stepper motor amp
34	PWMX	O	X axis pulse output for input to stepper motor amp
35	ISO OUT GND	O	Isolated Output Ground
36	+VCC	O	+ 5 volts
37	AMPENW	O	W axis amplifier enable
38	AMPENZ	O	Z axis amplifier enable
39	AMPENY	O	Y axis amplifier enable
40	AMPENX	O	X axis amplifier enable
41	LSCOM	I	Limit Switch Common
42	HOMEW	I	W axis home input
43	RLSW	I	W axis reverse limit switch input
44	FLSW	I	W axis forward limit switch input
45	HOMEZ	I	Z axis home input
46	RLSZ	I	Z axis reverse limit switch input
47	FLSZ	I	Z axis forward limit switch input
48	HOMEY	I	Y axis home input
49	RLSY	I	Y axis reverse limit switch input
50	FLSY	I	Y axis forward limit switch input
51	HOMEX	I	X axis home input
52	RLSX	I	X axis reverse limit switch input
53	FLSX	I	X axis forward limit switch input
54	+VCC		+ 5 volts
55	GND		Signal Ground
56	INCOM	I	Input common (Common for general inputs and Abort input)

57	XLATCH	I	Input 1 (Used for X axis latch input)
58	YLATCH	I	Input 2 (Used for Y axis latch input)
59	ZLATCH	I	Input 3 (Used for Z axis latch input)
60	WLATCH	I	Input 4 (Used for W axis latch input)
61	IN5	I	Input 5
62	IN6	I	Input 6
63	IN7	I	Input 7
64	IN8	I	Input 8
65	ABORT	I	Abort Input
66	OUT1	O	Output 1
67	OUT2	O	Output 2
68	OUT3	O	Output 3
69	OUT4	O	Output 4
70	OUT5	O	Output 5
71	OUT6	O	Output 6
72	OUT7	O	Output 7
73	OUT8	O	Output 8
74	GND		Signal Ground
75	AN1	I	Analog Input 1
76	AN2	I	Analog Input 2
77	AN3	I	Analog Input 3
78	AN4	I	Analog Input 4
79	AN5	I	Analog Input 5
80	AN6	I	Analog Input 6
81	AN7	I	Analog Input 7
82	AN8	I	Analog Input 8
83	+MAX	I	X Main encoder A+
84	-MAX	I	X Main encoder A-
85	+MBX	I	X Main encoder B+
86	-MBX	I	X Main encoder B-
87	+INX	I	X Main encoder Index +
88	-INX	I	X Main encoder Index -
89	ANA GND		Analog Ground
90	+VCC		+ 5 volts
91	+MAY	I	Y Main encoder A+
92	-MAY	I	Y Main encoder A-
93	+MBY	I	Y Main encoder B+
94	-MBY	I	Y Main encoder B-
95	+INY	I	Y Main encoder Index +
96	-INY	I	Y Main encoder Index -
97	+MAZ	I	Z Main encoder A+
98	-MAZ	I	Z Main encoder A-
99	+MBZ	I	Z Main encoder B+

100	-MBZ	I	Z Main encoder B-
101	+INZ	I	Z Main encoder Index +
102	-INZ	I	Z Main encoder Index -
103	GND		Signal Ground
104	+VCC		+ 5 volts
105	+MAW	I	W Main encoder A+
106	-MAW	I	W Main encoder A-
107	+MBW	I	W Main encoder B+
108	-MBW	I	W Main encoder B-
109	+INW	I	W Main encoder Index +
110	-INW	I	W Main encoder Index -
111	+12V		+12 volts
112	-12V		-12 volts

ICM-1900 Drawing



A4. AMP-19520/40

Description

The AMP-19540 (four-axis) and AMP-19520 (two-axis) are multi-axis brush/brushless amplifiers that are capable of handling 500 watts of continuous power per axis. Each amplifier is rated for 7 Amps continuous, 10 Amps peak and accepts 18-80 VDC for input power. Each AMP-195x0 includes a breakout for all controller I/O, including encoders, Hall-effect switches, digital inputs, digital outputs, and analog inputs.

The AMP-19520/19540Brushless drive module can be connected to any of the following Galil controllers via the 100-pin high density connector:

Accelera Series: DMC-42x0, DMC-18x6

Please refer to the AMP-19520/19540 User Manual for more information:

http://galilmc.com/download/manual/man_195x0.pdf

A5. CB-50-100 Adapter Board

Description

The CB-50-100 adapter board can be used to convert the CABLE-100 to (2) 50 Pin Ribbon Cables. The 50 Pin Ribbon Cables provide a versatile method of accessing the controller signals without the use of a Galil Interconnect Module.

Connectors

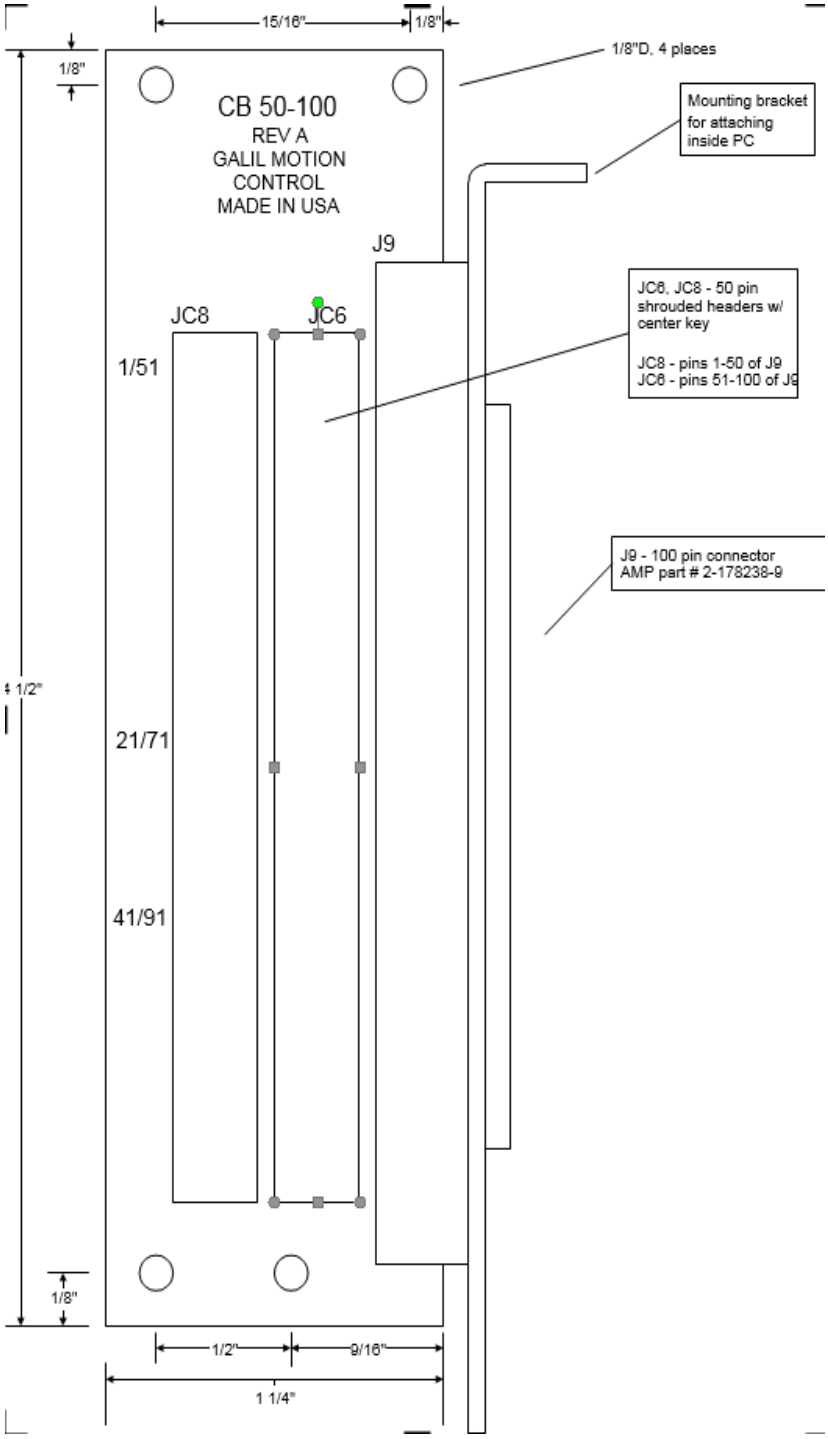
JC8 50 PIN IDC	J9 100 PIN HIGH DENSITY CONNECTOR
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28

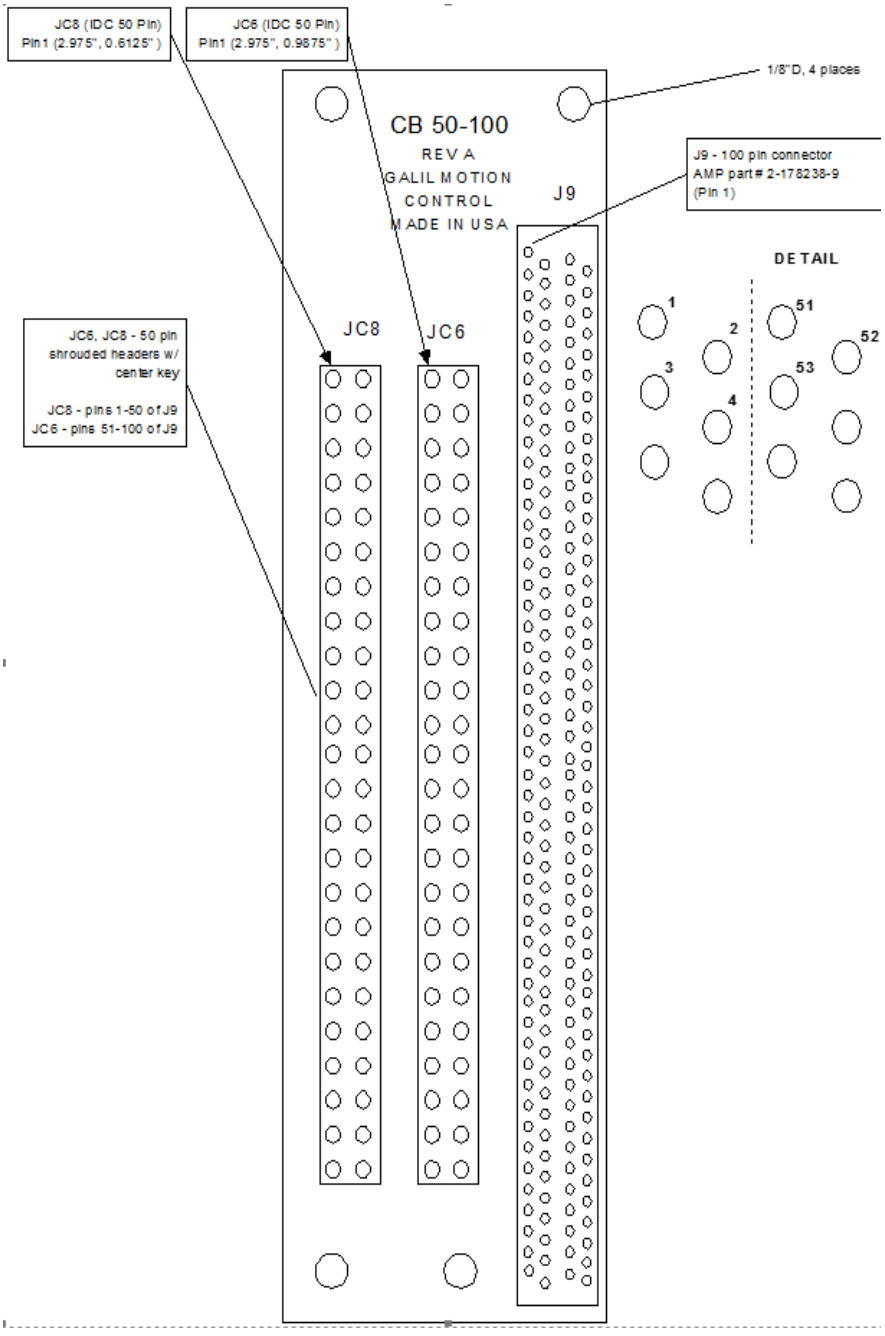
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50

JC6 50 PIN IDC	J9 100 PIN HIGH DENSITY CONNECTOR
1	51
2	52
3	53
4	54
5	55
6	56
7	57
8	58
9	59
10	60
11	61
12	62
13	63
14	64

15	65
16	66
17	67
18	68
19	69
20	70
21	71
22	72
23	73
24	74
25	75
26	76
27	77
28	78
29	79
30	80
31	81
32	82
33	83
34	84
35	85
36	86
37	87
38	88
39	89
40	90
41	91
42	92
43	93
44	94
45	95
46	96
47	97
48	98
49	99
50	100

CB-50-100 Drawings




JC8 (IDC 50 Pin)
Pin1 (2.975", 0.6125")
JC6 (IDC 50 Pin)
Pin1 (2.975", 0.9875")
1/8"D, 4 places
CB 50-100
REV A
GALIL MOTION
CONTROL
MADE IN USA
J9
J9 - 100 pin connector
AMP part# 2-178238-9
(Pin 1)
DETAIL
1
2
3
4
51
52
53
JC8
JC6
JC6, JC8 - 50 pin
shrouded headers w/
center key
JC8 - pins 1-50 of J9
JC6 - pins 51-100 of J9

A6. CB-50-80 Adapter Board

Description

The CB-50-80 adapter board can be used to convert the CABLE-80 to (2) 50 Pin Ribbon Cables. The 50 Pin Ribbon Cables provide a versatile method of accessing the extended I/O signals.

Connectors

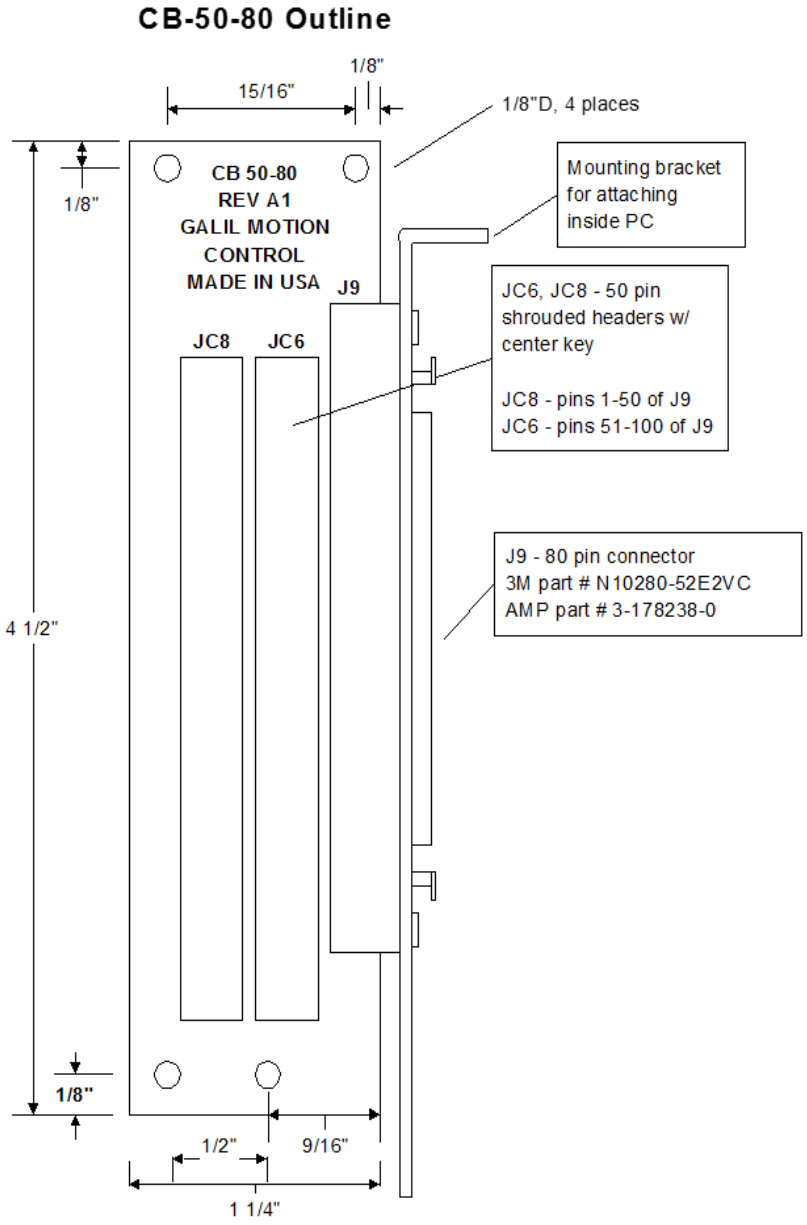
JC8 and JC6: 50 Pin Male IDC

J9: 80 Pin High Density Connector, AMP PART #3-178238-0

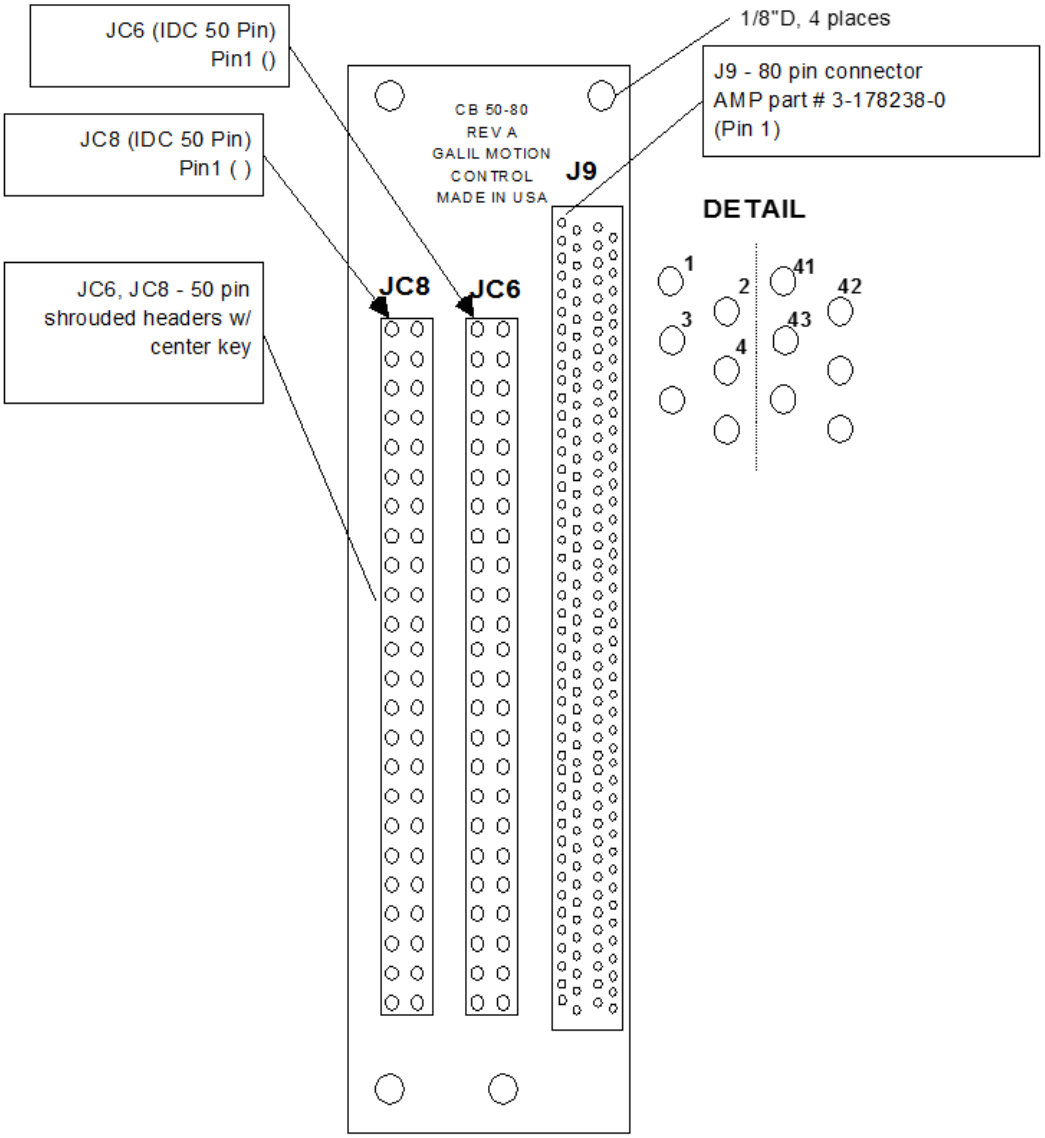
JC8 Pin	J9 Pin	JC8 Pin	J9 Pin
1	1	26	GND
2	2	27	27
3	3	28	GND
4	4	29	29
5	5	30	GND
6	6	31	31
7	7	32	GND
8	8	33	32
9	9	34	GND
10	10	35	33
11	11	36	GND
12	12	37	34
13	13	38	GND
14	14	39	35
15	15	40	GND
16	16	41	36
17	17	42	GND
18	GND	43	37
19	19	44	GND
20	GND	45	38
21	21	46	GND
22	GND	47	39
23	23	48	GND
24	GND	49	+5V
25	25	50	GND

JC6 Pin	J9 Pin	JC6 Pin	J9 Pin
1	41	26	GND
2	42	27	67
3	43	28	GND
4	44	29	69
5	45	30	GND
6	46	31	71
7	47	32	GND
8	48	33	72
9	49	34	GND
10	50	35	73
11	51	36	GND
12	52	37	74
13	53	38	GND
14	54	39	75
15	55	40	GND
16	56	41	76
17	57	42	GND
18	GND	43	77
19	59	44	GND
20	GND	45	78
21	61	46	GND
22	GND	47	79
23	63	48	GND
24	GND	49	+5V
25	65	50	GND

CB-50-80 Drawing



CB-50-80 Layout



List of Other Publications

"Step by Step Design of Motion Control Systems"

by Dr. Jacob Tal

"Motion Control Applications"

by Dr. Jacob Tal

"Motion Control by Microprocessors"

by Dr. Jacob Tal

Training Seminars

Galil, a leader in motion control with over 500,000 controllers working worldwide, has a proud reputation for anticipating and setting the trends in motion control. Galil understands your need to keep abreast with these trends in order to remain resourceful and competitive. Through a series of seminars and workshops held over the past 20 years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set-from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30 pm)

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

Those who consider themselves a "servo specialist" and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of “Motion Control Made Easy" or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8:00 am-5:00 pm)

PRODUCT WORKSHOP

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil’s headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a “hands-on” seminar and students can test their application on actual hardware and review it with Galil specialists.

Attendees must have a current application and recently purchased a Galil controller to attend this course.

TIME: Two days (8:30-4:30pm)

Contacting Us

Galil Motion Control

270 Technology Way

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-Mail Address: support@galilmc.com

Web: 370Hhttp://www.galilmc.com/

WARRANTY

All controllers manufactured by Galil Motion Control are warranted against defects in materials and workmanship for a period of 18 months after shipment. Motors, and Power supplies are warranted for 1 year. Extended warranties are available.

In the event of any defects in materials or workmanship, Galil Motion Control will, at its sole option, repair or replace the defective product covered by this warranty without charge. To obtain warranty service, the defective product must be returned within 30 days of the expiration of the applicable warranty period to Galil Motion Control, properly packaged and with transportation and insurance prepaid. We will reship at our expense only to destinations in the United States and for products within warranty.

Call Galil to receive a Return Materials Authorization (RMA) number prior to returning product to Galil.

Any defect in materials or workmanship determined by Galil Motion Control to be attributable to customer alteration, modification, negligence or misuse is not covered by this warranty.

EXCEPT AS SET FORTH ABOVE, GALIL MOTION CONTROL WILL MAKE NO WARRANTIES EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO SUCH PRODUCTS, AND SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

COPYRIGHT (3-97)

The software code contained in this Galil product is protected by copyright and must not be reproduced or disassembled in any form without prior written consent of Galil Motion Control, Inc.